Transactor

TRANSACTOR: THE MAGAZINE FOR COMMODORE PROGRAMMERS JANUARY 1988: VOLUME 8, ISSUE 4

Canada $4.25
USA $3.50

Computer-Generated Holography on the C64

Circles, Elipses and Polygons in ML

Projector Update adds Hidden Line Removal

Butterfield explains ML Square Roots

Supporting More Disk Formats under CP/M +

CP/M I/O Redirection

Reviews:

Turbo Processor for C64
Book of ML Routines
Merlin C128 Assembler

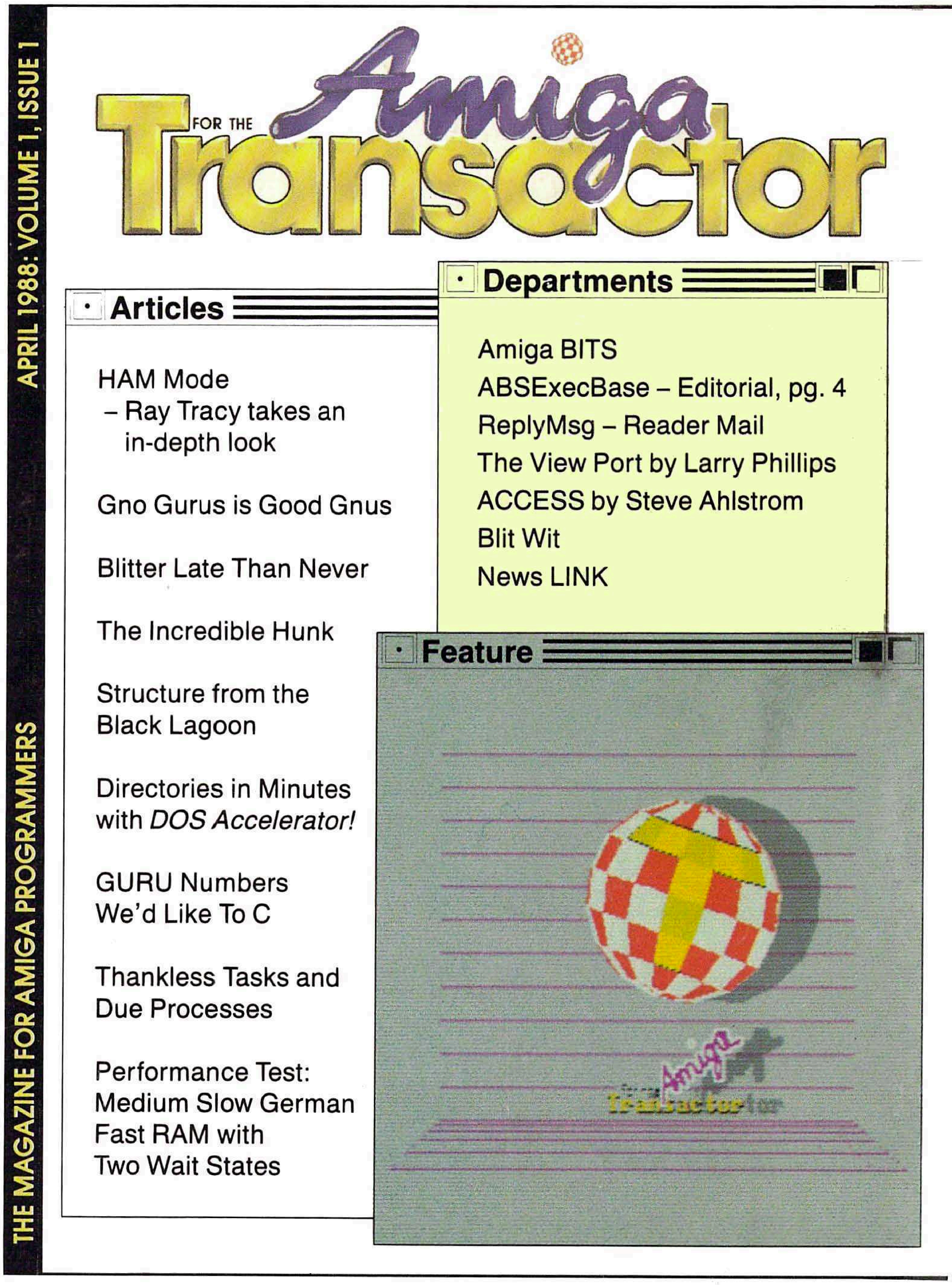
Amiga Section:

Tiny Window Manager

Two New Columns

Amiga Dispatches

Benchmark Modula-2



THE MAGAZINE FOR AMIGA PROGRAMMERS APRIL 1988: VOLUME 1, ISSUE 1

FOR THE Amiga Transactor

Articles

HAM Mode
– Ray Tracy takes an in-depth look

Gno Gurus is Good Gnus

Blitter Late Than Never

The Incredible Hunk

Structure from the Black Lagoon

Directories in Minutes with *DOS Accelerator!*

GURU Numbers We'd Like To C

Thankless Tasks and Due Processes

Performance Test: Medium Slow German Fast RAM with Two Wait States

Departments

Amiga BITS
ABSExecBase – Editorial, pg. 4
ReplyMsg – Reader Mail
The View Port by Larry Phillips
ACCESS by Steve Ahlstrom
Blit Wit
News LINK

Feature

The Potpourri Disk

Help!

This HELPful utility gives you instant menu-driven access to text files at the touch of a key – while any program is running!

Loan Helper

How much is that loan really going to cost you? Which interest rate can you afford? With Loan Helper, the answers are as close as your friendly 64!

Keyboard

Learning how to play the piano? This handy educational program makes it easy and fun to learn the notes on the keyboard.

Filedump

Examine your disk files FAST with this machine language utility. Handles six formats, including hex, decimal, CBM and true ASCII, WordPro and SpeedScript.

Anagrams

Anagrams lets you unscramble words for crossword puzzles and the like. The program uses a recursive ML subroutine for maximum speed and efficiency.

Life

A FAST machine language version of mathematician John Horton Conway's classic simulation. Set up your own 'colonies' and watch them grow!

War Balloons

Shoot down those evil Nazi War Balloons with your handy Acme Cannon! Don't let them get away!

Von Googol

At last! The mad philosopher, Helga von Googol, brings her own brand of wisdom to the small screen! If this is 'AI', then it just ain't natural!

News

Save the money you spend on those supermarket tabloids – this program will generate equally convincing headline copy – for free!

Wrd

The ultimate in easy-to-use data base programs. WRD lets you quickly and simply create, examine and edit just about any data. Comes with sample file.

Quiz

Trivia fanatics and students alike will have fun with this program, which gives you multiple choice tests on material you have entered with the WRD program.

AHA! Lander

AHA!'s great lunar lander program. Use either joystick or keyboard to compete against yourself or up to 8 other players. Watch out for space mines!

Bag the Elves

A cute little arcade-style game; capture the elves in the bag as quickly as you can – but don't get the good elf!

Blackjack

The most flexible blackjack simulation you'll find anywhere. Set up your favourite rule variations for doubling, surrendering and splitting the deck.

File Compare

Which of those two files you just created is the most recent version? With this great utility you'll never be left wondering.

Ghoul Dogs

Arcade maniacs look out! You'll need all your dexterity to handle this wicked joystick-buster! These mad dog-monsters from space are not for novices!

Octagons

Just the thing for you Mensa types. Octagons is a challenging puzzle of the mind. Four levels of play, and a tough 'memory' variation for real experts!

Backstreets

A nifty arcade game, 100% machine language, that helps you learn the typewriter keyboard while you play! Unlike any typing program you've seen!

All the above programs, just $17.95 US, $19.95 Canadian. No, not EACH of the above programs, ALL of the above programs, on a single disk, accessed independently or from a menu, with built-in menu-driven help and fast-loader.

The ENTIRE POTPOURRI COLLECTION JUST $17.95 US!!

See Order Card at Center

Transactor

Volume 8
Issue 4

Bits and Pieces .. 6
ML Break
Verify Bug
Sneaky File Print
Interrupt Routine Management
Auto-Linefeed Generation
80-Column Tricks
Simple Rules for the 128
1571 Seek-Stopper
C128 I/O Incompatibility
Function Key Finagler
Cursor Save and Restore
CLI Hint
AmigaDOS Rollodex Tool
AmigaBasic Fade-In

Letters 12
Advertising In The Transactor WORKS!
CP/M In The T.
. . .if I owned one I would get an Amiga magazine!
Program Listings For All Machines
Transactor Continuing Education Course
GEOS mice-and-little-pictures environment
Commodore 1526 Blues
Macro Assembler Desired

News BRK 76
Transactor for the Amiga
Half Price Until January 1st!
I Want To Switch!
I Want Both!
Subscription Renewals and Enquiries
Mail Order Products:
The 20/20 Deal
Transactor Special Offer
Transactor Mail Order
World Of Commodore Show
New England 1987 Computer Fair
RUN 64 Emulator for Amiga
Commodore Creating Product List
New UEDIT Release
Comspec Hard Drive for Amiga 1000
ROMDISK with HYPERBOOT for the C64
MIDI Interface for C64/128 and 64C
GEOS Upgrade for the C128
Video Digitizer for the C64
C128 CP/M Kit
Aegis "Art Paks" for the Amiga
Genealogy Program for the 8032
8051 Cross Assembler for C64/128
Oxxi Claims Benchmark M2

TransBloops ... 15
GAP FILL
Garbage Collector Revealed
Inside View
C128 Programmer's Aid Fix
Switchable RS-232 Interface
Getting Around With Gogo Dancer
Adding Analog RGB to the 1902

Start Address Announcing Transactor for the Amiga! 3
TeleColumn 16
The Projector Part II A new version with hidden line removal – and more 18
Computer-Generated Holography Explore a fascinating technology on your C64 29
Circles for the C64 Easy answers to a tough programming problem 37
Inside C128 CP/M Add support for virtually any CP/M disk format! 43
CP/M 3.0:Plus Redirection Diverting input and output with GET and PUT . 48
Square Roots in ML Jim Butterfield tells how – without floating point! 52
Placeholder for the C64 Short cursor save and restore routines 54

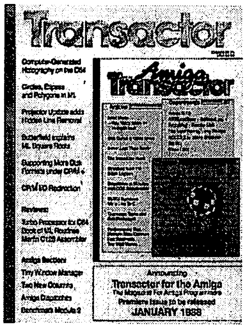
Reviews

1. Turbo Processor 65C816-based expansion hardware for the C64 with 64K RAM 56
2. ML Routines for the C64/128 new book from COMPUTE! 57
3. Merlin-128 A 6502 macro assembler development system 58
4. Benchmark Modula-2 A Modula-2 development system for the Amiga .. 59

Amiga Section

Amiga Dispatches Jerry explains why Ol' Myron is still his favourite joystick ... 63
Tiny Window Manager Clean up that Tiny Window Litter! 70
The View Port Larry Phillips looks at a hot IBM innovation - multitasking! 62
Access Steve Ahlstrom looks at the best Amiga freely distributable software 66

Note: before entering programs, see "Verifizer" on page 4


Transactor
Computer-Generated Holography on the C64
Circles, Ellipses and Polygons in ML
Projector Update adds Hidden Line Removal
Butterfield explains ML Square Roots
Supporting More Disk Formats under CP/M+
CP/M I/O Redirection
Reviews:
Turbo Processor for C64
Book of ML Routines
Merlin C128 Assembler
Amiga Section:
Tiny Window Manager
Two New Columns
Amiga Dispatches
Benchmark Modula 2
Amiga Transactor
Announcing
Transactor for the Amiga
The Magazine For Amiga Programmers
Premiere Issue to be released
JANUARY 1988

ABOUT THE COVER: The simulated Workbench windows were done with the same typesetting equipment that is used to produce the rest of the magazine – a Quadex 5000 typesetting system with a Compugraphic 8400 phototypesetter. The colour picture in the lower window was done on the Amiga by capturing the image from the famous "boing" demo (using "zsaveiff" from Meridian Software's Zing! package) and editing it with Deluxe Paint II. The photo was produced using a Polaroid Pallette system and "Imprint", an interface and program for the Amiga from American Liquid Light, Inc. The Polaroid Pallette has an internal CRT and exposes the film by displaying different parts of the screen image in black and white through a series of coloured filters. The Imprint software loads IFF picture files and controls the Polaroid Pallette to take the pictures (using standard 35mm film), and gives many options to the user for controlling the process. Thanks to Commodore Canada for the use of the Polaroid Pallette and Imprint equipment.

Transactor
The Magazine for Commodore Programmers

Editor-in-Chief
Karl J. H. Hildon

Publisher
Richard Evers

Technical Editor
Chris Zamara

Submissions Editor
Nick Sullivan

Customer Service
Jennifer Reddy

Contributing Writers

Ian Adam
Steve Ahlstrom
David Archibald
Anthony Bertram
Paul Blair
Tim Bolbach
Neal Bridges
Anthony Bryant
Jim Butterfield
Dale A. Castello
Betty Clay
Joseph Caffrey
Tom K. Collopy
Robert V. Davis
Elizabeth Deal
Frank E. DiGioia
Chris Dunn
Michael J. Erskine
Jack Farrah
Mark Farris
Jim Frost
Miklos Garamszeghy
Eric Germain
Michael T. Graham
Eric Guiguere
Thomas Gurley
R. James de Graff
Tim Grantham
Patrick Hawley
Adam Herst
Thomas Henry
John Houghton
Robert Huehn
David Jankowski
Clifton Karnes
Lorne Klassen
Jesse Knight
Gregory Knox
David Lathrop
James A. Lisowski
Richard Lucas
Scott Maclean
Steve McCrystal
Stacy McInnis
Chris Miller
Terry Montgomery
Ralph Morrill
D.J. Morris
Michael Mossman
Bryce Nesbitt
Gerald Neufeld
Noel Nyman
Matthew Palcic
Richard Perrit
Larry Phillips
Terry Pridham
Raymond Quirling
Doug Resenbeck
Richard Richmond
John W. Ross
Dan Schein
E.J. Schmahl
David Shiloh
Darren J. Spruyt
Aubrey Stanley
David Stidolph
Richard Stringer
Anton Treuenfels
Audrys Vilkas
Jack Weaver
Geoffrey Welch
Evan Williams

Production
Attic Typesetting Ltd.

Printing
Printed in Canada by
MacLean Hunter Printing

Program Listings In Transactor

All programs listed in Transactor will appear as they would on your screen in Upper/Lower case mode. To clarify two potential character mix-ups, zeroes will appear as '0' and the letter "o" will of course be in lower case. Secondly, the lower case L ('l') is a straight line as opposed to the number 1 which has an angled top.

Many programs will contain reverse video characters that represent cursor movements, colours, or function keys. These will also be shown exactly as they would appear on your screen, but they're listed here for reference. Also remember: CTRL-q within quotes is identical to a Cursor Down, et al.

Occasionally programs will contain lines that show consecutive spaces. Often the number of spaces you insert will not be critical to correct operation of the program. When it is, the required number of spaces will be shown. For example:

```
print "          flush right "   – would be shown as –   print "[10 spaces]flush right "
```

Cursor Characters For PET / CBM / VIC / 64

Down	q	Insert	T
Up	Q	Delete	t
Right	]	Clear Scrn	S
Left	[Lft]	Home	s
RVS	r	STOP	c
RVS Off	R		

Colour Characters For VIC / 64

Black	P	Orange	A
White	e	Brown	U
Red	£	Lt. Red	V
Cyan	[Cyn]	Grey 1	W
Purple	[Pur]	Grey 2	X
Green	↑	Lt. Green	Y
Blue	←	Lt. Blue	Z
Yellow	[Yel]	Grey 3	[Gr3]

Function Keys For VIC / 64

F1	E	F5	G
F2	I	F6	K
F3	F	F7	H
F4	J	F8	L

Please Note: Transactor's phone number is: (416) 764-5273

CompuServe Accounts

Contact us anytime on GO CBMPRG, GO CBMCOM, or EasyPlex at:

Karl J.H. Hildon	76703,4242
Chris Zamara	76703,4245
Nick Sullivan	76703,4353

Transactor is published bi-monthly by Transactor Publishing Inc., 85 West Wilmot Street, Unit 10, Richmond Hill, Ontario, L4B 1K7. Canadian Second Class mail registration number **6342.** USPS **725-050,** Second Class postage paid at Buffalo, NY, for U.S. subscribers. U.S. Postmasters: send address changes to Transactor, P.O. Box 338, Station C, Buffalo, NY, 14209 **ISSN# 0827-2530.**

Transactor is in no way connected with Commodore Business Machines Ltd. or Commodore Incorporated. Commodore and Commodore product names (PET, CBM, VIC, 64, 128, Amiga) are registered trademarks of Commodore Inc.

Subscriptions:
Canada $19 Cdn. U.S.A. $15 US. All other $21 US.
Air Mail (Overseas only) $40 US. ($4.15 postage/issue)

Send all subscriptions to: Transactor, Subscriptions Department, 85 West Wilmot Street, Unit 10, Richmond Hill, Ontario, Canada, L4B 1K7, 416 764 5273. Note: Subscriptions are handled at this address ONLY. Subscriptions sent to our Buffalo address (above) will be forwarded to our Richmond Hill HQ. For best results, use postage paid card at center of magazine.

Editorial contributions are always welcome. Minimum remuneration is $40 per printed page. Preferred media are 1541, 2031, 4040, 8050, 8250, 1571, or 1581 diskettes with WordPro, PaperClip, Pocket Writer, WordCraft, Superscript, (actually, just about any word processor files) or SEQ text files, or Amiga format 3½ diskettes with ASCII text files. Program listings including BITS subsmissions of more than a few lines should be provided on disk. Manuscripts should be typewritten, double spaced, with special characters or formats clearly marked. Photos should be glossy black and white prints. Illustrations should be on white paper with black ink only.

Quantity Orders:

In Canada:
Ingram Software Ltd.
141 Adesso Drive
Concord, Ontario
L4K 2W7
(416) 738-1700

In the U.S.A.:
IPD (International Periodical Distributors)
11760-B Sorrento Valley Road
San Diego, California
92121 (619) 481-5928
Ask for Dave Bruescher

SOLD OUT: The Best of The Transactor Volumes 1 & 2 & 3; Vol 4 Issues 03, 04, 05, 06, and Vol 5 Issues 02, 03, 04 are available on microfiche only
Still Available: Vol. 4: 01, 02. Vol. 5: 01, 04, 05, 06. Vol. 6: 01, 02, 03, 04, 05, 06.
Vol. 7: 01, 02, 03, 04, 05, 06. Vol. 8: 01, 02, 03, 04

Back Issues: $4.50 each. Order all back issues from Richmond Hill HQ.

Start Address

Transactor for the AMIGA

Inching Toward The Magazine Rack

Like asking a girl out for the first time, we're getting back into retail distribution. Cautiously, and after much rehearsal, we have started our approach... an approach that comes only after several months of investigating the mechanics of that stationary fixture known as "the magazine rack".

For those who subscribed based on our landmark decision to go off the news stands, please don't feel tricked. At the time, we had to do it – we simply couldn't afford to continue. And we are by no means getting back into "news stand" distribution. We won't be available at the grocery store, the local variety store or smoke shop, the hotel confectionery, or in train stations and airports. It's these places that are truly defined as "news stand". A print run to cover the shelf space in all those places would be four or five times our immediate objective and any attempt to tackle such an objective would send us spiralling right back out of the distribution business to square 1.

Computer shops and book stores, on the other hand, are known among publishers as "single-copy" outlets. Our single-copy sales always came in at around 100%, and it was the news stand sales that ruined our average. Time after time in city after city the story we always hear is, "By the time I get there, Transactor is sold out". But "there" was always *this* book store or *that* computer store. We could be wrong, but it's become apparent to us that Transactor readers generally don't go to the news stand to get their copy and, although it's not impossible that Transactor will show up at one or two of the millions of this type of outlet, it's highly unlikely that the distribution we're seeking will include them as a premeditated target.

We are, however, actively seeking pathways to the single-copy type outlets that did so well for us before: Waldenbooks, B. Dalton stores, Crown Books, Encore Books, Software Etcetera and ComputerLand; and in Canada W.H. Smith, Classic Bookstores, Coles and Lichtman's. We also want to ship to every computer shop on the planet that carries Commodore equipment.

We've made arrangements so far with a couple of magazine distributors, and any retailer you know is more than welcome to contact them directly. Their names are opposite on page 2. If neither is within reasonable geographic proximity, call us. By the time this edition reaches you we may have more regional distributors that we can recommend.

Some readers have told us that they wouldn't subscribe even if they couldn't get Transactor any other way. And unfortunately for us they've kept their word. But even more unfortunate is the fact that this page 3 will not reach them, except, perhaps, by accident. Because even more important than this development is the following news so many have been anxiously awaiting.

Announcing Transactor for the Amiga!

AmiEXPO in New York was the most electric computer show I've been to since the first World of Commodore show in Toronto. No shortage of adrenalin there – stories of "all-nighters" for last minute preparations were common conversation. Exhibitors and attendees came from as far as California, Vancouver, Great Britain and Germany. One report had attendance figures at 4,000 for the first day. I had my doubts at first, but when I stopped to consider that the seminars were filled to capacity while at the same time the exhibit floor was packed tight, 4,000 seemed like a conservative estimate.

We made the trip for a couple of reasons. Just seeing the show was well worth the effort, but spreading the word about "Transactor for the Amiga" brought overwhelming reactions from everyone. If only we'd had subscription forms with us! Even the staff of other magazines exhibiting at the show agreed that a high-tech journal for the Amiga was sorely needed and that they were glad to see us producing one – wonder if they really meant it.

If you own a modem, chances are you're already aware of this news. We've made similar announcements on just about every popular online service in North America including CompuServe, PeopleLink, BIX, GEnie, Quantum Link and PunterNet.

To elaborate, Transactor for the Amiga will not be at all unlike the original Transactor. We intend to publish articles of interest to programmers and hobbyists with an appetite for fat-free information that doesn't leave you feeling hungry again a short time after digesting it. A sample of what will appear come January is on the cover of this issue; however, don't take the article titles seriously, as if that need be said. We have lots of material lined up but didn't know if it would all appear in the premiere issue or if some would end up in the second issue. So we took the opportunity to have some fun instead.

By the way, if you have an idea for an article, please get in touch. We talked to several authors at the show and gave out a number of writer kits. Some already had articles assembled... articles that were declined by some of the other magazines because they were "too technical". In our opinion, there's no such thing as "too technical", so if you're in a similar situation, or know someone who is, send us your stuff or give us a call. I know when I invest time in a project that gets shelved for one reason or another, and then find a use for that work later on, it's like getting something for nothing. If the work gets published, then that "something" usually turns into money – and that's nice!

A charter subscription offer will be in effect until January 1, 1988... an offer that will never be repeated. It comes to HALF the regular price and a whopping 64% off the cover price. See News BRK and the sub card for more. We've addressed many reader comments about the advantages of buying at the magazine rack such as getting damage free copies and a cover without an ugly mailing label stuck on it. Starting with this issue, all "T" AND "T-A" subscribers will receive their copies in what our printer calls "a poly bag" – a supermarket type word for a piece of plastic that's hermetically sealed on three sides around the mag. Regardless, it should mean that if any damage occurs, you'll be able to toss the damage AND the label right into the trash, leaving a fresh, crisp, good-as-store-bought Transactor in hand.

Not mentioned anywhere else in this issue are our two incredible advertising offers also in effect until January 1 only. Ad rates in *Transactor for the Amiga* will for now be the same as in the original *Transactor*. But advertisers who take out space in the premiere issue of T-A will want to know about: 1) The Deep Discount Deal – a full 50% off ads in our first issue, and 2) The Double Exposure Deal – two ads for the price of one. Place any ad in the December issue of *Transactor* and get the same size ad in the Premiere Issue of *Transactor for the Amiga*, ABSOLUTELY FREE! At first glance this appears to be "1 for 5, 2 for 10" logic, but under plan 2, ads will reach Christmas shoppers. Again, if you're interested, or know someone who is, please call us soon. Although Christmas is still a while away, our deadlines are rapidly approaching and the pressman waits for nobody!

Lastly, some of you may be recalling previous editorials where I cite enormous work loads coupled with unbearable schedules and saying, "you guys must be crazy – as if one mag wasn't enough, now you'll have two!". Well, we've always been a little crazy, and if you think an occupational environment like that might suit you, we're currently accepting resumés.

Karl J.H. Hildon, Editor in Chief

Using "VERIFIZER"

The Transactor's Foolproof Program Entry Method

VERIFIZER should be run before typing in any long program from the pages of The Transactor. It will let you check your work line by line as you enter the program, and catch frustrating typing errors. The VERIFIZER concept works by displaying a two-letter code for each program line which you can check against the corresponding code in the program listing.

There are five versions of VERIFIZER here; one for PET/CBMs, VIC or C64, Plus 4, C128, and B128. Enter the applicable program and RUN it. If you get a data or checksum error, re-check the program and keep trying until all goes well. You should SAVE the program, since you'll want to use it every time you enter one of our programs. Once you've RUN the loader, remember to enter NEW to purge BASIC text space. Then turn VERIFIZER on with:

```
SYS 634 to enable the PET/CBM version  (off: SYS 637)
SYS 828  to enable the C64/VIC version  (off: SYS 831)
SYS 4096 to enable the Plus 4 version   (off: SYS 4099)
SYS 3072,1 to enable the C128 version  (off: SYS 3072,0)
BANK 15: SYS 1024 for B128  (off: BANK 15: SYS 1027)
```

Once VERIFIZER is on, every time you press RETURN on a program line a two-letter report code will appear on the top left of the screen in reverse field. Note that these letters are in uppercase and will appear as graphics characters unless you are in upper/lowercase mode (press shift/Commodore on C64/VIC).

Note: If a report code is missing (or "--") it means we've edited that line at the last minute which changes the report code. However, this will only happen occasionally and usually only on REM statements.

With VERIFIZER on, just enter the program from the magazine normally, checking each report code after you press RETURN on a line. If the code doesn't match up with the letters printed in the box beside the listing, you can re-check and correct the line, then try again. If you wish, you can LIST a range of lines, then type RETURN over each in succession while checking the report codes as they appear. Once the program has been properly entered, be sure to turn VERIFIZER off with the SYS indicated above before you do anything else.

VERIFIZER will catch transposition errors like POKE 52381,0 instead of POKE 53281,0. However, VERIFIZER uses a "weighted checksum technique" that can be fooled if you try hard enough; transposing two sets of 4 characters will produce the same report code but this should never happen short of deliberately (verifizer could have been designed to be more complex, but the report codes would need to be longer, and using it would be more trouble than checking code manually). VERIFIZER ignores spaces, so you may add or omit spaces from the listed program at will (providing you don't split up keywords!). Standard keyword abbreviations (like nE instead of next) will not affect the VERIFIZER report code.

Technical info: VIC/C64 VERIFIZER resides in the cassette buffer, so if you're using a datasette be aware that tape operations can be dangerous to its health. As far as compatibility with other utilities goes, VERIFIZER shouldn't cause any problems since it works through the BASIC warm-start link and jumps to the original destination of the link after it's finished. When disabled, it restores the link to its original contents.

PET/CBM VERIFIZER (BASIC 2.0 or 4.0)

```
CI  10 rem* data loader for "verifizer 4.0" *
CF  15 rem pet version
LI  20 cs=0
HC  30 for i=634 to 754:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
OG  60 if cs<>15580 then print"***** data error *****": end
JO  70 rem sys 634
AF  80 end
IN  100 :
ON  1000 data  76, 138,   2, 120, 173, 163,   2, 133, 144
IB  1010 data 173, 164,   2, 133, 145,  88,  96, 120, 165
CK  1020 data 145, 201,   2, 240,  16, 141, 164,   2, 165
EB  1030 data 144, 141, 163,   2, 169, 165, 133, 144, 169
HE  1040 data   2, 133, 145,  88,  96,  85, 228, 165, 217
OI  1050 data 201,  13, 208,  62, 165, 167, 208,  58, 173
JB  1060 data 254,   1, 133, 251, 162,   0, 134, 253, 189
PA  1070 data   0,   2, 168, 201,  32, 240,  15, 230, 253
HE  1080 data 165, 253,  41,   3, 133, 254,  32, 236,   2
EL  1090 data 198, 254,  16, 249, 232, 152, 208, 229, 165
LA  1100 data 251,  41,  15,  24, 105, 193, 141,   0, 128
KI  1110 data 165, 251,  74,  74,  74,  74,  24, 105, 193
EB  1120 data 141,   1, 128, 108, 163,   2, 152,  24, 101
DM  1130 data 251, 133, 251,  96
```

VIC/C64 VERIFIZER

```
KE  10 rem* data loader for "verifizer" *
JF  15 rem vic/64 version
LI  20 cs=0
BE  30 for i=828 to 958:read a:poke i,a
DH  40 cs=cs+a:next i
GK  50 :
FH  60 if cs<>14755 then print"***** data error *****": end
KP  70 rem sys 828
AF  80 end
IN  100 :
EC  1000 data  76,  74,   3, 165, 251, 141,   2,   3, 165
EP  1010 data 252, 141,   3,   3,  96, 173,   3,   3, 201
OC  1020 data   3, 240,  17, 133, 252, 173,   2,   3, 133
MN  1030 data 251, 169,  99, 141,   2,   3, 169,   3, 141
MG  1040 data   3,   3,  96, 173, 254,   1, 133,  89, 162
DM  1050 data   0, 160,   0, 189,   0,   2, 240,  22, 201
CA  1060 data  32, 240,  15, 133,  91, 200, 152,  41,   3
NG  1070 data 133,  90,  32, 183,   3, 198,  90,  16, 249
OK  1080 data 232, 208, 229,  56,  32, 240, 255, 169,  19
AN  1090 data  32, 210, 255, 169,  18,  32, 210, 255, 165
GH  1100 data  89,  41,  15,  24, 105,  97,  32, 210, 255
JC  1110 data 165,  89,  74,  74,  74,  74,  24, 105,  97
EP  1120 data  32, 210, 255, 169, 146,  32, 210, 255,  24
MH  1130 data  32, 240, 255, 108, 251,   0, 165,  91,  24
BH  1140 data 101,  89, 133,  89,  96
```

VIC/64 Double Verifizer **Steven Walley, Sunnymead, CA**

When using 'VERIFIZER' with some TVs, the upper left corner of the screen is cut off, hiding the verifizer-displayed codes. DOUBLE VERIFIZER solves that problem by showing the two-letter verifizer code on both the first and second row of the TV screen. Just run the below program once the regular Verifizer is activated.

```
KM  100 for ad=679 to 720:read da:poke ad,da:next ad
BC  110 sys 679: print: print
DI  120 print"double verifizer activated":new
GD  130 data 120, 169, 180, 141,  20,   3
IN  140 data 169,   2, 141,  21,   3,  88
EN  150 data 96,  162,   0, 189,   0, 216
KG  160 data 157,  40, 216, 232, 224,   2
KO  170 data 208, 245, 162,   0, 189,   0
FM  180 data   4, 157,  40,   4, 232, 224
LP  190 data   2, 208, 245,  76,  49, 234
```

VERIFIZER For Tape Users

Tom Potts, Rowley, MA

The following modifications to the Verifizer loader will allow VIC and 64 owners with Datasettes to use the Verifizer directly (without the loader). After running the new loader, you'll have a special copy of the Verifizer program which can be loaded from tape without disrupting the program in memory. Make the following additions and changes to the VIC/ 64 VERIFIZER loader:

```
NB  30    for i=850 to 980: read a: poke i,a
AL  60    if cs<>14821 then print"*****data error*****": end
IB  70    rem sys850 on, sys853 off
--  80    delete line
--  100   delete line
OC  1000 data  76,  96,   3, 165, 251, 141,   2,   3, 165
MO  1030 data 251, 169, 121, 141,   2,   3, 169,   3, 141
EG  1070 data 133,  90,  32, 205,   3, 198,  90,  16, 249
BD  2000 a$= "verifizer.sys850[space]"
KH  2010 for i=850 to 980
GL  2020 a$=a$+chr$(peek(i)): next
DC  2030 open 1,1,1,a$: close 1
IP  2040 end
```

Now RUN, pressing PLAY and RECORD when prompted to do so (use a rewound tape for easy future access). To use the special Verifizer that has just been created, first load the program you wish to verify or review into your computer from either tape or disk. Next insert the tape created above and be sure that it is rewound. Then enter in direct mode: OPEN1:CLOSE1. Press PLAY when prompted by the computer, and wait while the special Verifizer loads into the tape buffer. Once loaded, the screen will show FOUND VERIFIZER.SYS850. To activate, enter SYS 850 (not the 828 as in the original program). To de-activate, use SYS 853.

If you are going to use tape to SAVE a program, you must de-activate (SYS 853) since VERIFIZER moves some of the internal pointers used during a SAVE operation. Attempting a SAVE without turning off VERIFIZER first will usually result in a crash. If you wish to use VERIFIZER again after using the tape, you'll have to reload it with the OPEN1:CLOSE1 commands.

Plus 4 VERIFIZER

```
NI  1000 rem * data loader for "verifizer +4"
PM  1010 rem * commodore plus/4 version
EE  1020 graphic 1: scnclr: graphic 0: rem make room for code
NH  1030 cs=0
JI  1040 for j=4096 to 4216: read x: poke j,x: ch=ch+x: next
AP  1050 if ch<>13146 then print "checksum error": stop
NP  1060 print "sys 4096: rem to enable"
JC  1070 print "sys 4099: rem to disable"
ID  1080 end
PL  1090 data  76,  14,  16, 165, 211, 141,   2,   3
CA  1100 data 165, 212, 141,   3,   3,  96, 173,   3
OD  1110 data   3, 201,  16, 240,  17, 133, 212, 173
LP  1120 data   2,   3, 133, 211, 169,  39, 141,   2
EK  1130 data   3, 169,  16, 141,   3,   3,  96, 165
DI  1140 data  20, 133, 208, 162,   0, 160,   0, 189
LK  1150 data   0,   2, 201,  48, 144,   7, 201,  58
GJ  1160 data 176,   3, 232, 208, 242, 189,   0,   2
DN  1170 data 240,  22, 201,  32, 240,  15, 133, 210
GJ  1180 data 200, 152,  41,   3, 133, 209,  32, 113
CB  1190 data  16, 198, 209,  16, 249, 232, 208, 229
CB  1200 data 165, 208,  41,  15,  24, 105, 193, 141
PE  1210 data   0,  12, 165, 208,  74,  74,  74,  74
DO  1220 data  24, 105, 193, 141,   1,  12, 108, 211
BA  1230 data   0, 165, 210,  24, 101, 208, 133, 208
BG  1240 data  96
```

C128 VERIFIZER (40 column mode)

```
PK  1000 rem * data loader for "verifizer c128"
AK  1010 rem * commodore c128 version
JK  1020 rem * use in 40 column mode only!
NH  1030 cs=0
OG  1040 for j=3072 to 3214: read x: poke j,x: ch=ch+x: next
JP  1050 if ch<>17860 then print "checksum error": stop
MP  1060 print "sys 3072,1: rem to enable"
AG  1070 print "sys 3072,0: rem to disable"
ID  1080 end
GF  1090 data 208,  11, 165, 253, 141,   2,   3, 165
MG  1100 data 254, 141,   3,   3,  96, 173,   3,   3
HE  1110 data 201,  12, 240,  17, 133, 254, 173,   2
LM  1120 data   3, 133, 253, 169,  38, 141,   2,   3
JA  1130 data 169,  12, 141,   3,   3,  96, 165,  22
EI  1140 data 133, 250, 162,   0, 160,   0, 189,   0
KJ  1150 data   2, 201,  48, 144,   7, 201,  58, 176
DH  1160 data   3, 232, 208, 242, 189,   0,   2, 240
JM  1170 data  22, 201,  32, 240,  15, 133, 252, 200
KG  1180 data 152,  41,   3, 133, 251,  32, 135,  12
EF  1190 data 198, 251,  16, 249, 232, 208, 229,  56
CG  1200 data  32, 240, 255, 169,  19,  32, 210, 255
EC  1210 data 169,  18,  32, 210, 255, 165, 250,  41
AC  1220 data  15,  24, 105, 193,  32, 210, 255, 165
JA  1230 data 250,  74,  74,  74,  74,  24, 105, 193
CC  1240 data  32, 210, 255, 169, 146,  32, 210, 255
BO  1250 data  24,  32, 240, 255, 108, 253,   0, 165
PD  1260 data 252,  24, 101, 250, 133, 250,  96
```

B128 VERIFIZER

Elizabeth Deal, Malvern, PA

```
1 rem save"@0:verifizerb128",8
10 rem* data loader for "verifizer b128" *
20 cs=0
30 bank 15:for i=1024 to 1163:read a:poke i,a
40 cs=cs+a:next i
50 if cs<>16828 then print"** data error **": end
60 rem bank 15: sys 1024
70 end
1000 data  76,  14,   4, 165, 251, 141, 130,   2, 165, 252
1010 data 141, 131,   2,  96, 173, 130,   2, 201,  39, 240
1020 data  17, 133, 251, 173, 131,   2, 133, 252, 169,  39
1030 data 141, 130,   2, 169,   4, 141, 131,   2,  96, 165
1040 data   1,  72, 162,   1, 134,   1, 202, 165,  27, 133
1050 data 233,  32, 118,   4, 234, 177, 136, 240,  22, 201
1060 data  32, 240,  15, 133, 235, 232, 138,  41,   3, 133
1070 data 234,  32, 110,   4, 198, 234,  16, 249, 200, 208
1080 data 230, 165, 233,  41,  15,  24, 105, 193, 141,   0
1090 data 208, 165, 233,  74,  74,  74,  74,  24, 105, 193
1100 data 141,   1, 208,  24, 104, 133,   1, 108, 251,   0
1110 data 165, 235,  24, 101, 233, 133, 233,  96, 165, 136
1120 data 164, 137, 133, 133, 132, 134,  32,  38, 186,  24
1130 data  32,  78, 141, 165, 133,  56, 229, 136, 168,  96
1140 data 170, 170, 170, 170
```

b i t s

Got an interesting programming tip, short routine, or an unknown bit of Commodore trivia? Send it in – if we use it in the Bits column, we'll credit you in the column and send you a free one–year's subscription to The Transactor

ML Break

Amir Michail
Willowdale, Ontario

"ML Break" is a debugging tool that lets you cleanly exit from a machine language program by hitting the RESTORE key. What's more, the program counter (the address of the next instruction to be executed) is printed to the screen so you can see what the program was doing when you stopped it. Very handy for finding out where a program is getting hung up, and as a cleaner alternative to RUNSTOP/RESTORE.

Since the RESTORE key generates an NMI (Non–Maskable Interrupt), which "ML break" intercepts, you can get out of just about any crash.

```
JM  100 rem "ml break" by amir michail
NO  110 for j=710 to j+49
PE  120 read a: c=c+a: poke j,a: next
MN  130 if c<>5626 then print"data error!":stop
MH  140 poke 792,198: poke 793,2
KN  150 print"hit 'restore' to break"
EB  160 :
EJ  170 data 169, 247, 162,   2, 141,  24,   3, 142
EE  180 data  25,   3,  32, 225, 255, 208,   3,  76
JF  190 data  71, 254, 169,  42,  32, 210, 255, 104
GA  200 data 104, 170, 104,  32, 205, 189, 169, 164
PE  210 data  72, 169, 116,  72, 169,  32,  72, 169
LA  220 data 198, 162,   2, 141,  24,   3, 142,  25
DN  230 data   3,  64
```

Source code for ML Break

```
LH  100 nmivec  =    792        ;system vector
IL  110 basentr =    $a474      ;basic entry point
DO  130 break   lda  #<back     ;disable interupt
MP  140         ldx  #>back     ;by pointing vector
LG  150         sta  nmivec     ;to 'rti'
KE  160         stx  nmivec+1
BH  170         jsr  $ffe1      ;check stop key
OP  180         bne  break2     ;continue if not pressed
DB  190         jmp  $fe47      ;go to normal entry
HO  210 break2  lda  #42        ;print an asterisk
CM  220         jsr  $ffd2      ;in front of pc
JC  230         pla             ;discard status register value
PG  240         pla             ;low byte of pc
FM  250         tax
NK  260         pla             ;high byte of pc
LG  270         jsr  $bdcd      ;print pc
CA  280         lda  #>basentr  ;put entry point on stack
OJ  290         pha
GL  300         lda  #<basentr  ;so that the rti brings us there
CL  310         pha
MF  320         lda  #$20       ;cleared flags for status register
GM  330         pha
CE  340         lda  #<break    ;restore vector
NK  350         ldx  #>break    ;to this routine
EP  360         sta  nmivec
MB  370         stx  nmivec+1
IO  390 back    rti
```

Verify Bug

Kevin Hisel and Kevin Hopkins
Champaign, IL

We have uncovered a bug we have never seen mentioned before. It is with the VERIFY command in C64 and C128 BASIC (and presumably other versions as well).

During the development of a program that copies files, a little bug cropped up that consistently truncated the last block of the file. While this is all in a day's work for the plucky BASIC programmer/bug–hunter, its cause momentarily escaped us. Trying to figure out why our 55 block program was now 54, we VERIFYed the 55 block version (in memory) against the 54–block file. Of course, we knew that we would get the traditional "?VERIFY ERROR" but to our utter horror and disbelief, our seemingly undamaged machine reported, "OK".

It seems that VERIFY does not care if the file on disk is shorter than the program in memory. As long as the disk file matches memory in the machine up to the end of the file, VERIFY will *not* report an error. Admittedly, this condition will rarely occur, but here is how to duplicate it:

1. Save your favourite BASIC program to a spare disk.
2. Using your favourite sector editor (like Disk Doctor), truncate the file by writing a zero byte into the first position of a sector near the middle of the file. (This won't affect the block count in the directory unless you copy the file somewhere else.)
3. Load the original program into memory.
4. VERIFY it against the truncated version.

Your silly computer happily reports that the two programs are identical! But try to LOAD and RUN the truncated version and you will see that this is simply untrue!

Sneaky File Print — Amir Michail

To easily output a SEQ file to a printer:

```
load"filename,s",8
```

(hit RUN/STOP RESTORE if system hangs)

```
save"title"+chr$(13),4
```

That's it!

Interrupt Routine Management

Installing an IRQ-driven routine is not too difficult, but if you wish to be able to add new IRQ routines without disturbing current ones, and remove any individual IRQ routine, things can get messy. The following short routines make up an *interrupt handler* system that makes adding and removing IRQ routines (interrupt *servers*) simple and efficient.

There are four subroutines here: INSTHDLR, KILLHDLR, ADDSRVR, and REMSRVR. INSTHDLR is called once to install the IRQ handler system, and KILLHDLR is called to remove it before your program exits.

Once the handler is installed, adding a server to the interrupt chain is accomplished by simply calling ADDSRVR with the address of the subroutine to install in the accumulator (low) and y register (high). You will get back the number for that server in the x register; the server number is used to identify a server for removal.

To remove a server, just call REMSRVR with the server number in the x register.

You will appreciate the benefits of managing your interrupts with this system the first time you use it in a program. You never have to think about anything when you want extra work done during interrupts; just write the subroutine, and call ADDSRVR where you want it activated. When it comes time to shut it off, just REMSRVR it at the appropriate place.

It works by simply putting JSRs to the server subroutines in a chain that ends with a JMP to the original IRQ destination (usually, but not necessarily, the Kernal IRQ entry point at $EA31). Unused server slots in the chain are filled with three NOPs to take the place of a JSR. A server is added to the chain by having a JSR followed by its address placed in the first empty (NOP-filled) slot in the server chain, and removed by having the JSR simply NOPped out. Simple, perhaps, but it works like a charm.

The system as listed can handle up to four interrupt servers. This number can be changed to any size you need by changing the assembler variable IHENTRIES. Each additional server added to this maximum uses three more bytes in the server chain, and causes three extra NOPs to be executed every IRQ.

In case you haven't guessed, the inspiration for these routines comes from the Amiga's AddIntServer, RemIntServer, and SetIntHandler functions. Now you can program interrupts on your 64 just like they do on the Amiga (well, almost).

Interrupt Handler System

```
JN  1000 ; interrupt handler system
IG  1010 ;
PG  1020 ; insthdlr, killhdlr,
OB  1030 ; addsrvr, remsrvr
GI  1040 ;
AH  1050 tempvec=    $fb
KJ  1060 ;
JP  1070 ihentries=    4            ;max # of int servers
GE  1080 ; increase if necessary
IL  1090 ;
PK  1100 chainlen=    ihentries * 3
MM  1110 ;
GN  1120 ;
HA  1130 ihchain  *=*+chainlen
BK  1140 ; server chain goes here
KG  1150          rts
OP  1160 ;
IA  1170 ;
OP  1180 insthdlr  =    *
JO  1190 ; install the interrupt handler
IA  1200          ldy  #chainlen-1
AG  1210          lda  #$ea
CI  1220 ih1      sta  ihchain,y   ;put nops in
AC  1230          dey              ;handler chain
IO  1240          bpl  ih1
IF  1250 ;
EK  1260          php
PI  1270          sei
OF  1280          lda  $0314       ;save irq vector
GF  1290          sta  intexit+1
KB  1300          sta  saveirq
MJ  1310          lda  $0315
HH  1320          sta  intexit+2
EE  1330          sta  saveirq+1
CL  1340 ;
HJ  1350          lda  #<handler
LJ  1360          sta  $0314     ;point vector to handler
HK  1370          lda  #>handler
AC  1380          sta  $0315
CD  1390          plp
OO  1400 ;
OG  1410          rts
CA  1420 ;
MA  1430 ;
DN  1440 killhdlr  =    *
CM  1450 ; restore irqs as before insthdlr
MG  1460          php
HF  1470          sei
```

```
AJ  1480            lda  saveirq
GF  1490            sta  $0314      ;restore irq vector
AL  1500            lda  saveirq+1
CK  1510            sta  $0315
EL  1520            plp
GO  1530            rts
KH  1540 ;
EI  1550 ;
BM  1560 addsrvr =  *
PM  1570 ; adds routine at address .a/.y
EG  1580 ; to the interrupt chain;
KD  1590 ; returns the number of the
GF  1600 ; server in the chain in .x
AM  1610 ;
MA  1620            php
HP  1630            sei
ON  1640 ;
GM  1650            sta  tempvec
JA  1660            sty  tempvec+1
MP  1670 ;
JM  1680            lda  #$20       ;jsr
EP  1690            ldy  #chainlen-3
BP  1700            ldx  #3
EG  1710 ais1       cmp  ihchain,y  ;skip full slots
OM  1720            bne  ais2
NH  1730            dey
HI  1740            dey
BJ  1750            dey
HJ  1760            dex
OB  1770            bpl  ais1
LB  1780            bmi  ais3
HA  1790 ais2       sta  ihchain,y
CB  1800 ; put handler call in next
JD  1810 ; available slot
CD  1820            lda  tempvec
HI  1830            sta  ihchain+1,y
PB  1840            lda  tempvec+1
MJ  1850            sta  ihchain+2,y
EO  1860 ais3       plp
KD  1870            rts
OM  1880 ;
IN  1890 ;
EC  1900 remsrvr =  *
NI  1910 ; removes the interrupt server
NH  1920 ; specified in .x
JC  1930 ; from the interrupt chain
ME  1940            php
HD  1950            sei
OB  1960 ;
IC  1970            clc
HK  1980            lda  #0
JH  1990 ris1       dex
EA  2000            bmi  ris2
NL  2010            adc  #3
HA  2020            bne  ris1
KL  2030 ris2       tay
PD  2040            lda  #$ea       ;nop out jsr to server
AD  2050            sta  ihchain,y
NG  2060            sta  ihchain+1,y
```

```
IH  2070            sta  ihchain+2,y
EO  2080            plp
GB  2090            rts
KK  2100 ;
EL  2110 ;
EM  2120 handler =  *
BH  2130            jsr  ihchain    ;execute servers
MK  2140 intexit =  *
PI  2150            jmp  $ea31
BE  2160 ;($ea31 modified by insthdlr)
AP  2170 ;
ME  2180 saveirq *=*+2
NK  2190 ;original irq vector is saved here
```

Auto-Linefeed Generation

**Joseph Buckley
Quincy, MA**

Page 349 of the Commodore 64 Programmer's Reference Guide states that opening an RS-232 channel with a logical file number of greater than 127 forces an automatic linefeed after each carriage return. What may not be commonly known is that this only works when sending to the file using PRINT from BASIC. I came across this while using a printer that needed linefeeds.

What I finally did was place a short wedge in the CHROUT routine. I changed the vector IBASOUT at $0326 to point to this code.

```
wedge  cmp #$0d
       bne  skip
       jsr   $f1ca
       lda   #$0a
skip   jmp  $f1ca
```

This will add a linefeed after every carriage return printed. For a more general routine, you could limit it to device number two.

Easy User-Alert

Tom Morrow, Oak Park, IL

Often in a program it becomes necessary to signal to the user that something important is happening (like they are about to format a disk, or exit the program without saving work). The following short machine language routine performs this function quite economically. Include these lines at the beginning of your program:

```
10 data 172,  32, 208, 202, 142,  32, 208, 165
20 data 198, 240, 248, 140,  32, 208,  96
30 for ct=679 to 693: read dt: poke ct,dt: next ct
```

When you want to alert the user, just SYS 679. A pattern will form in the border of the screen until the user presses a key. After the keypress, the border will revert to its original colour.

A usage example:

```
10 print "do you really want to exit? (y/n)"
20 sys 679: get an$
```

C128 Bits

80–Column Tricks

Kevin Hisel
Champaign, IL

I stumbled across the following tricks with the VDC chip in the C128. Use them in 80–column mode while in BASIC BANK 15.

```
10 rem wild screen roundup
20 for i=6 to 80: sys 52684,i,1: next

10 rem character swipe
20 for i=0 to 8: sys 52684,i,23
30 for x=1 to 100: next x,i

10 rem open the curtain
20 for i=0 to 100: sys 52684,i,35
30 for x=1 to 10: next x,i
```

Simple Rules for the 128

Kevin Hisel

Being the librarian for a user group, I come across lots of 128 public domain programs. Almost always, these programs need to be slightly modified because the programmer did not follow a few simple and courteous rules. Here they are:

Rule #1: Always place a REM in the first few lines that states what machine the program runs on. Many frustrated 64 owners may get hold of your program and wonder for days why it won't run.

Rule #2: Always check to see what screen (40 or 80 column) the user is running with and if necessary, adjust your program or simply display a message telling the user to switch modes. It is very easy:

```
if rwindow(2)=80 then print"for 40 column mode only": end
```

This can save a lot of head–scratching when someone tries to RUN your latest 40 column hi–res demo or 80 column spreadsheet, etc.

Rule #3: If your program redefines the function keys, save the user's keys first, and when the program ends, *put them back!* Again this is very easy to do:

To save the keys:

```
for i=4096 to 4352: poke i+2048,peek(i): next
```

To put them back when you are done:

```
for i=4096 to 4352: poke i,peek(i+2048): next
```

These two operations take a few seconds to execute, but the user will be grateful that when they hit the F3 key for a directory next time, they won't format the disk or some such thing.

1571 Seek–Stopper

Gerald Boersma
Nepean, Ontario

On the C–128 with the 1571 disk drive, the drive does a lot of seeking on power–up if the disk within is single sided. In an earlier column, a reader mentioned that a solution is to copy all files to another disk, format the disk as double sided, and copy the files back again. However, there is an easier way! You can just format the second side of the disk while leaving side one alone.

Type:

```
open 15,8,15,"u0>m0"
print#15,"u0>h1"
```

This puts the 1571 drive in 1541 mode, then selects the second read/write head. At this point the drive error light may flash, since the second side of the disk is not formatted. Don't worry about it; you are about to format this side:

```
print#15,"n0:anti–knock,ak"
```

After the format takes place, the disk is prepared for future use. Now, when you boot up with that disk in the drive, the 1571 will immediately find both sides and won't chatter at you while it tries to make sense of the second side.

CAUTION: Don't use this on "flippies", disks that have data on both sides, since you will wipe out the second side.

C128 I/O Incompatibility

Richard Thornton
Richmond, Virginia

Recently I spent considerable time trying to convert a C64 machine language program to the C128. On the 128 version, the disk file it produced was incorrect, and some data was erratically written to the screen rather than the file. The problem turned out to be consecutive calls to the CHKOUT routine with no prior call to CLRCHN. Apparently, the C64/1541 has no problem with this, but the C128/1571 can't handle it properly. Preceding each CHKIN and CHKOUT with a call to CLRCHN solved the problem.

Function Key Finagler

Ed Schmahl
Bowie MD

Here is a way to make your C–128 function keys self–modifying, so that every time you hit a function key, numbers within the key definition will increase or decrease by a set value. This is useful, for example, when you are developing a program and wish to save it with a new version number each time. With "function key finagler", you could set up a function key like this:

```
key5,"dsave" + chr$(34) + "file.001"
   + chr$(34) + ":sys4864,5,1,2"
```

The first time you press F5, the file "file.001" will be saved; the next time, the name will be "file.002", etc.

The "sys4864" in the above example is what increments the value in the function key. You can put the routine anywhere in memory; it's fully relocatable. The syntax for using it is:

```
SYS ad,a,x,y
```

ad is the start address of the routine (4864 in the above example); a is the function key number (1 to 10); x is the amount of the increment (which can be negative using 255 for –1, 254 for –2, etc.); and y is the position within the function key definition, with zero being the position of the first occurence of a digit.

Here are a few other self-modifying function keys that you might find interesting:

```
key1,"graphic(0and1):sys4864,1,1,0"+chr$(13)
```
(toggles graphics modes)

```
key2,"vol5:sound2,4000,6:sys4864,2,3,11"+chr$(13)
```
(beep with varying pitch)

```
key3,"a=(00and15)+1:color4,a:sys4864,3,1,1"+chr$(13)
```
(change border colour)

```
key7,"list00000-00100:sys4864,7,1,2
:sys4864,7,1,8"+chr$(13)
```
(list forward one block at a time)

```
key8,"list00800-00900:sys4864,8,255,2
:sys4864,8,255,8"+chr$(13)
```
(list one block at a time backwards)

Change these key commands to suit your taste, or invent other ones as you like. Remember to insert enough leading digits to prevent wrap-arounds in the numbers. The ML program is only 80 bytes, so you can stuff it almost anywhere – even in the upper part of key memory ($1000–$10ff), if your key messages are short.

C128 Function Key Finagler

```
NI  100 rem function key finagler
EM  110 ad=4864: rem relocatable
AF  120 for i=0 to 79: read n
BD  130 poke ad+i,n: ck=ck+n: next
CO  140 if ck<>11227 then print"data error"
KA  150 :
BO  160 data 132, 250, 168, 169,  16, 133, 252, 169
JJ  170 data   0, 133, 251, 136,  24, 136,  48,   4
GL  180 data 113, 251, 208, 249, 105,   9, 133, 251
CG  190 data 200,  56, 177, 251, 233,  48, 201,  10
FB  200 data 144,   6, 230, 251, 208, 244, 240,  39
NM  210 data 164, 250, 138, 113, 251, 176,  15, 201
PF  220 data  58, 144,  26, 233,  10, 145, 251, 136
FP  230 data  48,  21, 169,   0, 240, 237, 201,  48
OF  240 data 176,  11, 105,  10, 145, 251, 136,  48
CC  250 data   6, 169, 255,  48, 222, 145, 251,  96
```

Cursor Save and Restore

Paul Blair
Canberra, Australia

This routine is one I've always wanted, and the C128 makes it easy. There I am, wanting to print something on the screen, jump off to another spot to print something else, then return to where I left off, to print more.

Basic 7 has two routines to help me do this, SAVEPOS and RSTRPOS at $CC1E and $C932 respectively. SAVEPOS saves the current cursor location (column and row) at $DE/$DF. RSTRPOS recovers them and puts you back whence you came. Try this (from bank 15):

```
1 scnclr: char 0,4,7,"this is up here"
2 sys 52254: sleep 2: char 0,11,20,"look!"
3 sys 51506: sleep 2: print " again"
```

Amiga Bits

CLI Hint

Steve Tibbett
Gloucester, Ontario

Here's a tip I find useful at times for starting a bunch of tasks without stuff going on while I'm typing: If you hold down the CTRL key when you hit RETURN after typing a CLI command, the CLI will not start processing that line until you hit the RETURN key alone. So, if you type LIST <CTRL–RETURN> DIR <RETURN>, LIST will only be executed after the final RETURN, followed by DIR.

This can come in handy when you want execute a few commands, but wish to wait until a large program has loaded to avoid making the disk head seek back and forth between files; while the program is loading, you can type all your commands using CTRL–RETURN between them, then just press the final RETURN when disk activity stops.

AmigaDOS Rollodex Tool

Benjamin Dobkin
Rego Park, NY

This short Execute file will retrieve all the "rollodex cards", from a properly formatted file, whose first line matches a supplied argument.

For example, to find "Jones" in your database file called "addresses", you would type from the CLI:

```
execute rollodex jones addresses
```

All entries whose first line started with "Jones" would be completely printed out.

The "rollodex" execute file is simply this:

```
.KEY string,database
echo >ram:temp "0(F BU/<string>/;P;TN);STOP"
edit from <database> to nil: with ram:temp OPT P6W60
delete >nil: ram:temp
```

It uses the line editor Edit (included on the standard Workbench disk), so that program must be in your C directory or somewhere else in the search path for Rollodex to work. Your "database" is just a standard ASCII file that you can make with a text editor (Ed, Emacs, etc.), with each entry taking a fixed number of lines, for example:

```
--------------------
Jones, Fred
18 Beazly Street
London, England
phone 212-304-3123

--------------------
Jane, Mary
14 41st St.
New York, NY
phone 501-123-4351

--------------------
```

In a file like this, each record is six lines long. This corresponds to the first 6 in the "OPT P6W60" option given to Edit in the execute file. You can change this to suit your needs.

Rollodex works by giving commands to Edit. It creates a temporary command file for Edit in RAM: that looks like this, with the argument <string> substituted for your search string:

```
"0(F BU/<string>/;P;TN);STOP"
```

This means go forward, look at the beginning of each line for the uppercase string, when it's found, back up to the previous line, then type to the end of the buffer. The brackets group these instructions together, and the 0 in front means repeat them indefinitely, until end of file. The STOP exits Edit and returns to the CLI.

Edit is then called, using the command file just created and the options P6W60, giving the buffer size (six lines of up to sixty characters each). As explained above, you can change the buffer length to suit your database file format.

Note that in the database file, the search string must start with an uppercase character and must begin in the first column. Also, the match is made by the first characters on the line, so searching for "SM" will match "Smith", "Smurf", "Smedley", etc. The "-----" used above is not necessary, but some form of delimiter between entries is a good idea for clarity.

Now you have a simple database, without spending money or typing in a long program!

AmigaBasic Fade–In

**Graham Reed
Toronto, Ontario**

This simple AmigaBASIC program will create a "fade–in" or "fade–out" effect by varying the intensities of all four colour registers from the given colours to black, or from black to the given colours. It currently does the first four colour registers only, but it could be easily modified to work with 8, 16, or 32–colour screens as well, although a visible "wash" would occur due to the speed of Basic.

The fade is accomplished by the subprogram *fadein*, which is passed arrays containing the the red, green, and blue colour intensities (from 0 to 1) for each of the four colours in the Workbench screen, and a fade direction. If the direction is nonzero, the routine fades the screen in from black to the specified colours; if zero, it fades out from the given colours to black.

The mainline part of the program shows how to use *fadein*: it loads colour values (specified here in their "real" form from 0 to 15 and converted to Basic's 0 to 1 representation) into arrays, and passes those arrays to *fadein* along with a direction. To show off, it keeps fading in and out until you click the left mouse button.

```
' fade in/out
main:
 DIM r(3), g(3), b(3)
 FOR i=0 TO 3 'read rgb values for each colour register
  READ r, g, b
  r(i)=r/15: g(i)=g/15: b(i)=b/15
 NEXT i
 PRINT "Click left mouse button to end"
 WHILE MOUSE(0)=0
  CALL fadein( r(), g(), b(), 0 )
  CALL fadein( r(), g(), b(), 1 )
 WEND
END

' r, g, b values for each colour register
DATA  0,  5, 10
DATA 15, 15, 15
DATA  0,  0,  2
DATA 15,  8,  0

SUB fadein( red(1), green(1), blue(1), dir%) STATIC
 IF dir% THEN
  dir% = 1: st = 0: en = 1 'fade in
 ELSE
  dir% = -1: st = 1: en = 0 'fade out
 END IF
 FOR i=st TO en STEP dir% * .0625
  FOR j%=0 TO 3
   PALETTE j%, i*red(j%), i*green(j%), i*blue(j%)
  NEXT j%
 NEXT i
END SUB
```

LeTteRs

"Advertising In The Transactor WORKS!": In your July '87 issue of the Transactor you published my press release in your NEWS BRK section. Immediately following publication we experienced a dramatic increase in BBS sales from customers both in Canada and the previously untapped U.S. market.

I would like to thank you for providing this wonderful service. Without help of this kind, small mail-order operations, like myself, would not survive. Pass the message to your prospective advertisers: advertising in the Transactor WORKS!

James MacFarlane, Co-Author, Spence XP BBS
Islington, Ontario, Canada

CP/M In The T: Thank goodness, somebody realises that CP/M articles are NEEDED. I have the C128, can't afford to buy an Amiga – and if I could my wife wouldn't stand for spending that much money on my hobby – so I have to explore all facets of the computer I have! I have dropped my subscriptions to Compute Gazette and RUN because they never publish articles on the CP/M mode.

Please encourage Adam Hearst and Clifton Karnes to keep writing . . . for your CP/M audience.

J. Nelson William, San Diego, California

At first we stayed away from CP/M because of its age; the duration of its existence led us to believe that there was already plenty of information available in other magazines and books. Turned out we were wrong – CP/M info has gone dry, and help can be hard to find. So we began actively look for a little CP/M material each issue. Thanks to Adam Herst, Clifton Karnes, Aubrey Stanley, Mike Garamszeghy and others, CP/M is shedding its years and looking good. Thanks for the support.

". . .if I owned one I would get an Amiga magazine!": Mr. Bernard H. Weiss (Letters, Volume 8, Issue 2) has expressed what has been on my mind for some time in regards to your Amiga coverage. There's too much of it! My 64 has done the job for me for 4 years and so has the 1541 (without trouble!). The Amiga is a great computer but if I owned one I would get an Amiga magazine. Computing is a hobby with me, but if I were to get on the treadmill of buying a new machines every year or so I could not afford the Transactor!

P.S. I do not like your Volume 8, Issue 3 cover! The old artwork was worth the price of the "T" by itself. Also, the paper is too shiny – hard on the eyes!!!

Hans Uechert, Wymark, Saskatchewan

Commodore has produced some pretty good computers over the years. And since 1978, we have covered them all, in one way or another. The Amiga is yet another Commodore computer, so we've covered it too. It's nice to work with, powerful and full of possibilities. But you're right – Amiga users should buy an Amiga-specific magazine. That's why, come early January, Transactor will undergo mitosis. You'll get your eight-bit journal once again. No more Amiga coverage! This cell-division, though, will spawn a second Transactor for Amiga users only – "Transactor for the Amiga". (Note to Amiga users looking for more details: see the editorial in this issue.)

You don't like our CN Tower photograph! John Mostacci would be glad to hear that you miss him. But the new cover design is here to stay – at least till the next time we change it. After painstaking market research, we have determined that white magazine covers show up better on a newsstand shelf that coloured ones do (actually, we just went to a magazine store and had a good look around, but it amounts to the same thing). Luckily for us, a lot of readers seem to approve of the new covers, though we have to admit there's quite a few disgruntled Duke fans out there. But what could we do? Duke got himself an AT clone, and he'll probably show up in one of the MS-DOS magazines any month now, reviewing productivity software.

Program Listings For All Machines: Your last news stand issue was the first issue I ever saw of the Transactor. I regret the past issues I missed, but I will be part of the future. Since my greatest pleasure is writing my own programs, your instructional listings in hex or Basic is what I am looking for.

I do have a request (that you must get all the time from owners of the C128). When my current program ran out of string space (especially variable space), I upgraded from the C64 to the C128. I noticed most of your articles deal with the C64. I suppose the philosophy in computerland is that since we have a C64 in our C128 we are covered. However, I bet that most of your readers are also hackers like myself and once we have put our programs into C128 format, we can't go back to the C64. For instance, "XREF for the 64" by David Archibald makes my mouth water. I could really use that help with my program. Since it is written with data statements I can't change it for my C128. My request is that you might make it a policy that any C64 article you print would be purchased (of course at a lower fee) in revised C128 format if anyone in your reading audience has the ability or patience to do so. I noticed that Compute! Magazine often lists the same program for several different computers. Maybe this means fewer programs per issue, but nobody is left biting their finger nails (except C128 owners in Compute! magazine).

P.S. It would also make purchasing your disks more reasonable. At this point it would be just another frustration.

Cynthia Darrow, Watertown, NY

That's an interesting idea, Cynthia. In fact, whenever possible, we do like to run multiple versions of our published programs. Also, we usually are able to publish the source for machine language programs (though not in the case of XREF, unfortunately!); we do this for the very reason that it enables readers to modify the code to suit their own needs or their own machines. If readers wish to send us ported versions of programs from the magazine, we will certainly consider them for publication (or perhaps for the disk, depending on the size of the program, and so on). In any case, good ideas don't just materialize once then vanish, so you'll more than likely see an original C128 cross-reference program in Transactor some time in the near future.

Transactor Continuing Education Course: In the course of a long and varied academic career, now defunct by a good ten years, it has been my lot to deal with Continuing Education courses offered by our University – in fact, I still operate several of them in the field of Microbiology and Public Health.

As a result of this experience and the fact that it seems unduly torturous to learn 6502 assembly language in any short time, solo, it occurs to me that, given a sufficiently large, interested body of eager neophytes, you and your considerable bevy of top-notch experts could either offer such a course by mail for an appropriate fee, or it could be activated as a continuing series in Transactor, my all time favourite mag.

One might suppose that you have thought through such a plan many times and discarded it for any number of arcane reasons. Even if so, perhaps another look might produce different results.

Bob Tischer, Starkville, MS

Bob, we have never considered providing a course through the mail before. With our small staff, it's difficult enough to just keep producing the magazine. You could be right, though. If sufficient interest did materialize, I am sure we could round up a host of top-name authors to give us a hand with the series. Food for thought. Maybe your letter will generate some mail for us. How about it folks – Continuing Education, by mail, in intermediate to advanced programming skills? Comments, anyone?

"GEOS mice-and-little-pictures environment": GEOS doesn't seem to get much respect among "techie" users. While I can understand that, I have noticed that it's often the people who are most enamored of the Amiga mice-and-little-pictures environment who sneer the most at the GEOS mice-and-little-pictures environment. Since the T is the foremost in Commodore techie magazines, I'd be most interested in seeing a few articles about GEOS, pro or con (and I'm aware that the "pro" ones will be hard to come by!) Any possibility of that?

Marte Brengle, Burbank, California

GEOS. It's been a tough decision to make since day one. GEOS is great for those users who find it suits their style, but so far it doesn't seem to have been inspiring for many programmers. If we're going to cover GEOS, it will be from the programmer's point of view – our normal slant, in other words. So far, we haven't been getting those articles and, frankly, we wouldn't know where to look for them outside the walls of Berkeley Softworks. And so, we're silent on GEOS... not from principle, but for lack of material. Sorry Marte.

Commodore 1526 Blues: I am one of the poor people who bought a Commodore 1526/MPS 802 printer. I say poor, although I have been quite pleased with it as far as it goes. The major drawback with it (as you probably well know) is its very, very limited graphics capability. I have often thought that, given its reputation as a "smart" Commodore peripheral, it should be possible to create a new ROM chip for it which would make it work like am MPS 801 (1525). I would be interested in doing it myself except that my skills are all in software with only limited skills in the practical aspects of firmware and hardware. I'm afraid of frying my printer, computer and every other solid state electronic device in my house if I do anything more than take a chip or board out and put another in. Do you or any of your readers know of any companies or individuals who have done such a public service? Given that the printer seems to be modeled after the Mannesman Tally Spirit 80, I should think that it is quite possible. I believe that the Spirit 80 has graphics capability. In a similar vein, have you heard of interfaces that allow Commodore printers to be attached to other computers (e.g. you-know-who and compatibles) or are the CBM printers just too smart for that kind of dumb hookup?

Peter Chynoweth, Saskatoon, Saskatchewan

Prepare for a feast. Within the next few months, if things continue to go well, a new series of hardware modification kits will become available through Transactor for the 1526 printer. The mods are in three stages. First, a change of ROMs that will clean up all existing 1526 bugs, increase the printing speed, and also provide a whole new set of commands through a printer command channel. These commands include a way to tell the 1526 to emulate a 1525 printer flawlessly! Until the code is complete, I can't go any further on the command set.

The second mod kit provides a circuit board containing a new microprocessor and 1K of RAM that will replace the 1526's normal *processor. The purpose: to allow the 1526 to access more memory through a better chip, thereby allowing fonts to be downloaded into the additional RAM. The third mod kit contains a panel of switches to be mounted on the 1526. These switches will allow control of Top-Of-Form, On & Off-Line and whatever else can be thought of. Just be patient for a little while longer – the way it looks right now, the answer to your 1526 woes is right around the corner.*

Using An IEEE Disk Drive With The Commodore 64: I have always felt that it would be a real asset to have a dual disk drive. I was disappointed when I learned that the manufacturer of the MSD-2 was no longer in business. However, I was attracted by an advertisement for interfacing a Commodore 4040 dual disk drive with the C64 by using an RTC C64-LINK II. This interface also raised the BASIC 2.0 to 4.0 (16 additional disk commands).

I have had continuous problems with this arrangement. The BACKUP and COPY commands do not work if there is any kind of glitch on the source disk. Many programs such as PRINTSHOP and NEWSROOM wil not load from the 4040. By typing in SERIAL, I am able to default the system to the 1541 drive and thus load the programs. I do not regard this as being a satisfactory arrangement as the very reasons for acquiring a dual disk drive have been defeated. It would appear that there are inherent differences between the two drives but it escapes my logic as to why. I do know that a large number of programs can be loaded from both disk drives and formatted disks from either drive are compatible. Have any of your readers encountered this problem and, if so, have they been able to solve it? I have already invested considerable time and money in this system and would hate to abandon it now. Any help you can offer would be appreciated.

If a program uses disk protection or direct disk programming techniques, it should be stated on the label in BOLD PRINT. But it never is. If a program has been written that requires the 1541 to operate, such as Printshop or Newsroom, then chances are that you cannot use another drive in its place. Advanced programming techniques that exploit internal features of the 1541 disk drive cannot possibly be expected to work on any other disk drive. The 1541 and the 4040 differ in the amount of RAM they contain, the usage of that RAM, the code contained in ROM, plus the way in which the drives handle data. The 4040 is an IEEE-488 parallel drive. The 1541 is a pseudo-serial drive. By looking through the source code for each, similarities and differences abound. They share lineage, but are not the same drives. Enough said.

The 4040 drive was never meant to be a bit copier. If an error was encountered during a Backup or Copy, then the procedure would bomb out every time. The RTC Link cannot be expected to improve on the drive. One problem inherent with the RTC Link, though, was its consumption of RAM. In order for it to provide you with an IEEE-488 interface plus give those "16 additional disk commands", some liberties were taken. Incompatibility with some commercial programs will be found.

We have been using the entire line of IEEE-488 drives with our 64s for years. Karl right now has a 4040 and an 8250 plus several 8050s, all hooked together over our G-Link IEEE-488 interface for the 64. Although you don't get any extra commands, it does provide clean and fast access to the IEEE-488 bus. It's an ugly interface, but it works.

One last point before going on. MSD did not go out of business. They simply stopped supporting the Commodore marketplace. This rumour has been spreading for quite some time, and it's time that it stopped. The president of MSD called us about it one day after he read a somewhat exaggerated account of his company's demise in our NEWS BRK section. Needless to say, MSD is alive, well, and staying far away from the Commodore community.

Macro Assembler Desired: I am at my wits' end in dealing with my Commodore Macro Assembler package. I am fine until I attempt to assemble a macro. When the macro is expanded, the assembler gives me an error. Give me the names of a few good macro assemblers (make sure they include excellent documentation). I noticed that Transactor uses PAL. I am unfamiliar with PAL. Is it a MACRO assembler? Where can I get it? Any help that you can give me to get me started on the road to machine language proficiency would be greatly appreciated.

Jeffry S. Barnes, Clarksville, IN, USA

PAL is not a macro assembler. It is just a nice 6502 assembler to use with the PET/CBM, B128 and C64 microcomputers. Currently, it is being marketed by Spinnaker in the US under the "Better Working Software" label, along with POWER 64, as "The Programmers' Toolbox". Together, PAL and POWER make a terrific buy at any price. I'll provide Spinnaker's address, plus Spinnaker's Canadian distributor's address and such at the end of this reply.

The finest 6502 macro assembler that I have used to date is the POWER Assembler (aka Buddy 128 and Buddy 64), also distributed by Spinnaker. Buddy was written by Chris Miller, a name that will be familiar to many of our readers. Chris has been regularly writing for us for years. A few issues ago we published Chris' "Mr. Ed", a text-editor for the C64. You'll find a full-blown version of Mr. Ed on either Buddy disk. This text editor is part of the version of Buddy (EBUD) that will assemble text data files in RAM. You'll also find a version of Buddy (BUD) that will assembler PAL-type source files (i.e. Basic source), and a final version for C128 users that will assemble Z80 code on the C128 side. To make this package even better, Chris tossed on a slew of macro source files, extra utilities and whatever else he could think of. It was a true labour of love. And it is not copy-protected in the least.

The documentation is good. But that was to be expected considering that Chris Miller wrote the docs. The manual is informative, well laid out, and fun when the going gets tough. In short, he did a good job. Since the first time Chris sent me a copy, I have been passing it about to people that I knew would use it and advise Chris on how to make it better. Liz Deal, Mike Garamszeghy, Aubrey Stanley and I have caused Chris more than enough headaches, but to date, he has always come through. Buddy is perhaps the nicest and most versatile assembler that you'll find for the Commodore line of computers.

US Distributor	*Canadian Distributor*
Spinnaker	*Beamscope Canada Inc.*
One Kendal Street	*110 Commander Blvd.*
Cambridge, MA, 02139	*Scarborough, Ontario, Canada*
(617) 494-1200	*M1S 3H7 (416) 291-0000*
800-826-0706 toll-free	*800-268-3521 toll-free (Ontario only)*
	800-268-5535 toll-free (rest of Canada)

Tranzbloopurz

GAP FILL, Vol5, Issue6, page 57

This one goes back almost three years! Paul Blair of Holder, Australia, writes, *"Could I point out a very minor but important glitch in GAP FILL. The problem is in line 230. If the order of file opening is reversed, the program runs quite happily on my 4040. So, I suggest line 230 should read:*

```
230 OPEN 5,8,5,"#":OPEN 15,8,15: REM etc
```

If I don't do this, the error light flashes, the drive gets hysterical, and the program gets knotted up. Not a pretty sight.

I've never worked out quite why this order of things is required. Rae West, in his opus magnus on PET programming, gives it both ways – as you had it on page 185 (the introduction to use of disk drives) then uses the reverse order in all the working examples. There's obviously a reason, but I just don't know what it is. Maybe someone else does."

Ideas, anyone?

Garbage Collector Revealed, Vol8, Issue2, page 30

A problem that won't affect correct operation of the program, but an interesting one to look at. On lines 1140, 1150 and 1180 of the PAL source listing, the word "collect" in the comment field has been replaced by the word "input". This, as author Michael Graham quite correctly points out, is the result of tokenizing the source code on one machine, and listing it on another.

Inside View – bits, page 8

It turns out that this program will not work on 64s that have ROM version 2, because it stores to display memory without modifying the corresponding colour memory. If you have an older 64 and find that the program behaves strangely, that's probably the reason. Our apologies to anyone who typed it in and can't use it; unfortunately, it's not possible – and usually not necessary – for us to test everything on all ROM versions.

C128 Programmer's Aid Fix – Letters, page 17

After stating that the new version of C128 Programmer's Aid would be fixed and put on Transactor Disk #20, we found that the code changes were responsible for not only fixing problems, but creating some new ones too. If we get a good one working, we'll put it on a T disk and let you know.

Switchable RS-232 Interface, page 21:

The circuit diagram left out a few labels: the ICs, starting with the 1488 and moving clockwise, should be labelled U1 through U5; the diodes at the upper left of the diagram are, from top to bottom, D1, D2 and D3; the capacitors just to the right of the diodes should be labelled C1 and C2, again from top to bottom. Also, the 1N4001 diodes, as they are named in the diagram, are incorrectly referred to as 1N001 in the parts list. Another problem with the diagram: the small note at the bottom saying to ground all unused pins on the 1488 and 1489s is incorrect: only the 1488 should have its unused pins grounded; the unused pins on the 1489s should be left as N/C – no connection. Finally, the voltage ratings for the disk capacitors in the parts list are not specified – the ratings are unimportant, and the lowest you can get will do.

Getting Around With Gogo Dancer, page 47:

There were a few small bugs in the Gogo Dancer program; here is the revised BASIC loader. This new version arrived at HQ about 2 days after we went to press. Although the original works in most case, it will not process line labels and computed line numbers beyond a "THEN" statement. This new one does. The changes from the original listing are shown in boldface:

```
DP | 100 rem save"0:gogowedge.ldr",8
EP | 110 rem ** written by chris miller, kitchener, ontario
GJ | 120 rem
EJ | 130 for j=49152 to 49395 : read x
HA | 140 poke j,x : ch=ch+x : next
PE | 150 if ch<>31618 then print"checksum error" : end
OL | 160 rem
AO | 170 data 169,  11, 141,   8,   3, 169, 192, 141
GO | 180 data   9,   3,  96,  32, 115,   0,  32,  20
JF | 190 data 192,  76, 174, 167, 201, 139, 240,  15
BA | 200 data 201, 137, 240,  77, 201, 141, 240,  47
KA | 201 data201,  64, 208,  37,  76, 248, 168,  32
LM | 203 data115,   0,  32, 158, 173, 165,  97, 208
PI | 205 data   6,  32,   9, 169,  76, 251, 168,  32
KA | 207 data121,   0, 201, 137, 240,  43, 169, 167
IE | 209 data  32, 255, 174, 201, 128, 144,  37, 176
BC | 210 data203,  32, 124,   0,  76, 237
HE | 220 data 167, 169,   3,  32, 251, 163, 165, 123
EE | 230 data  72, 165, 122,  72, 165,  58,  72, 165
DC | 240 data  57,  72, 169, 141,  72,  32, 105, 192
PB | 250 data  76, 174, 167,  32, 115,   0, 201,  64
BA | 260 data 240,  12,  32, 124,   0,  32, 158, 173
JD | 270 data  32, 247, 183,  76, 163, 168,  32,   6
IG | 280 data 169, 136, 177, 122, 201,  36, 208,  34
EF | 290 data  32, 115,   0,  32, 139, 176, 160,   0
HE | 300 data 177,  71, 240,  91, 133, 255, 200, 177
CI | 310 data  71, 170, 200, 177,  71, 168, 138, 208
KH | 320 data   1, 136, 202, 134, 253, 132, 254,  76
EJ | 330 data 178, 192, 132, 255, 165, 122, 133, 253
IK | 340 data 165, 123, 133, 254, 165,  43, 133,  95
MK | 350 data 165,  44, 133,  96, 160,   4, 166, 255
HA | 360 data 177,  95, 201,  64, 208,  25, 136, 136
PF | 370 data 136, 177, 253, 200, 200, 200, 200, 209
HI | 380 data  95, 208,  12, 202, 208, 240, 200, 177
OJ | 390 data  95, 240,  23, 201,  58, 240,  19, 160
MJ | 400 data   0, 177,  95, 170, 200, 177,  95, 133
GA | 410 data  96, 134,  95, 177,  95, 208, 205,  76
CD | 420 data 227, 168,  56,  76, 197, 168
```

Adding Analog RGB Capability to the 1902 Monitor

Nothing wrong with this one, except that it might be a little hard to find using our Table of Contents. It's on page 72.

TeleColumn

Transactor Online Conference

Saturday, November 21 at 10:00 PM Eastern (7:00 Pacific)
in CompuServe's CBMCOM CO
An evening with

Ben Dunnington and Mark Brown of INFO Magazine

Online with Ben and Mark

If you're looking for info on a piece of Commodore related hardware or software, chances are good than Mark or Ben have it. I guess that's whay their magazine is called "INFO", and it's one of our favourites. INFO is pressing near 200,000 copies per issue now and has worldwide distribution.

INFO's claim to fame is "the first personal computer magazine produced entirely with personal computers". So if you're looking for some advice about desktop publishing, you'll want to talk to Ben and Mark – they've been at it since before the term was invented.

Since we're starting an Amiga only publication, we're also looking forward to asking them if they have any similar plans of their own.

So don't miss out! Sign on (use 300 baud - it's cheaper) and enter GO CBMCOM. At the main Function prompt enter "CO" – we'll use the default CO channel.

Don't Be Fooled

The ads for GEnie circulating in the major mags compare CompuServe's $39.95 registration fee against their $18.00 fee to sign up. The $39.95 charge will indeed get you registered for using CompuServe, but that also includes manuals, a subscription to Online Today (CompuServe's own monthly magazine) and $25.00 of paid usage time. I'm not sure what GEnie throws in for the 18 bucks, just as I'm equally unsure what The Source gives you for their $49.95. All figures in US dough, too!

However, CompuServe offers extensive online help at all prompts and for those with any telecomputing experience at all, the manuals quickly become unnecessary.

Want to save 40 bucks and get something for nothing at the same time? There's another way to get a CompuServe membership. It's called a "CompuServe Intro Pak". The cost? Absolutely FREE! And it comes with a $15.00 time credit so you can try it out for a while to see if you like it. All you need is a credit card.

Just send us a self addressed envelope (please, no stamps – save them for Christmas cards). We'll turn it around with an Intro Pak inside (one per person please). When you get it, you'll notice there's a "snap pack" at the center of the booklet. Inside you'll find an account number and a password. At the back of the book are phone numbers for hundreds of CompuServe network nodes, as well as numbers for the other data carriers such as Tymnet and Telenet. The inside back cover will show how to get connected to the service after you've established a connection with the node.

The "Host Name" is "CIS" – you may see this, probably not. Now enter your account number, hit RETURN, and follow with your password. CompuServe will notice that this is your first time on and ask you for a method of payment so have your credit card number handy. Cheque-Free is another payment method offered by CIS, but I believe this may require a validation period. Regardless, your first $15.00 of connect time is free – if you enter a credit card number you'll be sent straight into the system.

About two weeks later, CompuServe will send you a new password. This is to protect against unauthorized use of your credit card – the new password goes to the address that belongs to your card number. To change your password again, enter GO PASSWORD at any prompt.

Depending on the transmission speed you connect at, $15.00 will get you different amounts of free time. At 1200 bps, $15.00 will cover you for just over an hour; at 300 bps, $15 is good for about 2½ hours (does not include Telenet, Tymnet or DataPac charges). Also in the back of the Intro Pak are access instructions for hundreds of forums within the system. Enter GO CBMPRG or GO CBMCOM, for example, and you'll have navigated your way directly to the forums managed by us at the T. On your first entry to the forums you'll be presented with the Visitor's Menu, but please select the "Join" option – there's no extra charge for this, and by joining you'll be allowed to Leave messages and download programs from the Data Libraries. So Leave us a message – we check into the forums every day and we'll have a reply out to you usually within 24 hours.

The same ads mentioned above show GEnie's connect time charges to be less than The Source and CompuServe's. But remember, "you get what you pay for" and, based on reports I've received via Email, telecomputing services are no exception to this rule. Also, CompuServe is one of the only services with their own nationwide network. From most major metros, CIS is a local call and the network charge is 25 cents per hour. Most others can only be accessed using the independent data carriers, or by dialing long distance. Although Tymnet, Telenet and DataPac offer local numbers in most areas, using their services adds a surcharge that makes the basic hourly rate substantially higher.

We've had several reviews published in recent issues by users of the major telecomputing services, but other opinions are always welcome. I don't think GEnie, BIX or PeopleLink have been in the TeleColumn spotlight yet – maybe someone who uses several services would be in a good position to write 'em up... someone that could be dragged away from his Amiga for an hour or so... someone like... Tim Grantham perhaps (hint, hint).

Sunday Night COs with Nick Sullivan et al.

Care to chat online? Perhaps you're looking for a program that isn't in the Data Libraries. Maybe you've just bought something new with features not explained very well in the manual. Or you've been working on a program all weekend and you just can't seem to get past a stubborn bug. I could think of a hundred other reasons, but if any of them happen to coincide with Sunday night at 10 (or thereabouts), sign on, GO CBMPRG and enter the CO area. Nick Sullivan and any other sysops, assistant sysops, or just plain knowledgable types, will be making regular appearances in the conferencing area every Sunday at 10. By the way, use 300 baud – it's cheaper and you can't type at 120 characters per second anyway.

Attention Anchor 6480 Users

Thought you bought a lemon, did you? Most 6480 owners I've talked to have all but given up on the "black sheep" of modems for Commodore equipment. But there is hope. Word is that version 1.1 of AutoCom works pretty well. On CompuServe the C64 version has scrolling problems, but can be fixed by changing your terminal type to "CRT" in GO PROFILE.

To get a copy, call Anchor Automation at (818) 997-6493.

CBTerm Now Available in CBMCOM

One of the most popular and most powerful terminal programs for the C64 is now available in the CBMCOM forum. CBTerm features include Xmodem protocol transfer capabilities with automatic Image header stripping for downloading programs uploaded to CompuServe with B protocol. It has a software 80 column screen mode, with dual incoming and outgoing windows. There are dozens of support files and overlay modules available too, including an overlay to make CBTerm work with the Anchor 6480.

Don't have a program that will download using Xmodem protocol? A catch-22 situation because without such a program, you can't download CBTerm so that you can use it to download other programs. Not to worry, mate. Sign on to CBMCOM and we'll direct you to a program called BXD – Boot Xmodem Download. This short little BASIC program is easy to type in and can be used just long enough to get a more comprehensive terminal emulator... like CBTerm! Once you have CBTerm you discard BXD and enjoy your full-featured access to one of the most comprehensive collection of programs available online or anywhere.

Call Waiting Disable

Do you have call waiting? You know, that handy little telephone feature you pay extra for every month? It's great when you're on the phone a lot – when someone else calls, you hear a beep and put your first call on hold while you answer the second. That beep, however, can really mess up an online connection, as many have found out the hard way. Just before the beep, there's usually a split second of silence that a modem would consider as "no connection" – the carrier is lost for just long enough to make the call disconnect. When it happens, about all you can do is *answer your second call.*

Recognizing this problem, most telephone services are now offering "call-waiting disable". By dialing "*70", your call-waiting feature is temporarily disabled at the main switching station. You'll then hear a second dial tone, at which point you proceed with the number of the service you want to call. If you don't have touchtone, dialing "1170" works the same way. If neither works, try calling your business office for an answer, and then please share it with us.

On autodial Hayes and Hayes compatible modems, the command "ATDT*70,5551212" usually works, where "5551212" is, of course, the number of the service. If your modem doesn't accept the asterisk, use "ATDT1170,5551212". The comma gives a two second pause to wait for the second dial tone before proceeding with the number.

I've heard different reports on re-enabling your call-waiting feature. In the U.S., it seems to vary depending on the company servicing your calls. Some say you must dial *70 or 1170 to re-activate; others say it's automatically re-enabled when you hang up. In Canada it doesn't matter, because I can't get it to disable, let alone re-enable. When I called our local business office I was told there's no way to do what I'm describing, but again, the service may or may not be available in other areas – call the business office number on your phone bill to find out for sure.

Mitey Mo Programming Tips

Before we log off, here's a technical tidbit taken from a reply by Gary Farmaner to a question about the Mitey Mo modem on CBMPRG:

The Mitey Mo, in addition to hook, uses lines of the user port to control the carrier tone and the answer/originate mode. To make sure all the lines are set up properly, use POKE 56579,38. This is a universal poke; it works with every modem.

With the Mitey Mo modem, you deal with the following lines by reading or changing the following bits in location 56577 (bit 0 is LSB, bit 7 is MSB):

Hook	bit 5	zero/online	one/offline
Carrier	bit 4	zero/no carrier	one/carrier present
Ring	bit 3	zero/ringing	one/no ring
Tone	bit 2	zero/carrier tone	one/carrier tone off
Mode	bit 1	zero/answer mode	one/originate mode

(For a BBS, you'll want to be in answer mode; callers will use originate.)

For answering a call, you'd do the following: Wait for ring with bit 3; go online with bit 5; put on your carrier tone with bit 2; wait for carrier using bit 4.

It's a good idea to count down a number of occurences of a ring-detect before registering it as a ring. The ring detect line will alternate between zero and one throughout a ring, at the ring frequency. Occasionally a single ring transition will occur without an actual ring, so counting off a few transitions before accepting will ensure reliable ring detection.

The Projector Part II

Ian Adam
Vancouver, BC

. . . The most significant improvement is the treatment of hidden lines. . .

Part I of the Projector was published in Volume 6 Issue 4 of Transactor. It produces a three-dimensional plot of any mathematical formula on the screen of the Commodore 64, using Gary Kiziak's High-Res graphics utility from Volume 5, Issue 6. That program has now been improved through the addition of several new capabilities. It has also been extended to the new Commodore 128.

Hidden Lines

The most significant improvement to the routines is the treatment of hidden lines. The clarity of the plot is greatly enhanced by deleting lines that should be hidden from view by other parts of the subject. Achieving this required some modification to the original plotting routines, so you will have to obtain the revised machine code in order to see it in action on the 64.

Conceptually, removing hidden lines is not difficult, if one makes some assumptions about the solid shape being plotted. In this case, we assume that the shape is an upward-facing surface that does not double back on itself; that is, there are no caves and the bottom surface, if any, is not visible. With this assumption, objects in the background are only visible if they are taller than the foreground.

To implement this theory, a buffer of 320 bytes is reserved, each byte corresponding to one column of pixels on the screen. The algorithm uses the buffer to keep track of the highest point plotted so far. When the plot is first started, the buffer is cleared to all zeros. The plotting is then done from foreground to background. Each time a point is plotted, its height is compared to the appropriate byte in the buffer; if the point is higher than the value found, then the point is plotted and the buffer is updated by substituting the new height. If the current point is lower than the value in the buffer, the point is assumed to be obscured and is not plotted. The theory works well, and does not slow down the plotting perceptibly. In fact, if there were a lot of hidden points, it could possibly speed the process a little.

Two entry points to the routines are required in order to handle this:

```
SYS 49155,x,y [TO x,y . .]  draw a conventional line
SYS 49191,x,y [. . .      ]  draw line that may be hidden
```

Printer

The high-res plotting procedure creates some problems with printer dumps. The screen used is tucked away under the operating system ROM at E000. This is ideal for video, as it is free RAM that is easily accessed by the video chip. However, in order to read it, the CPU has to bank out the operating system, which makes the printer inaccessible. Part I of The Projector included a short piece of code to relocate the screen to $2000, where any printer dump can easily find it. This requires leaving The Projector after each plot, loading and running the relocate routine, then loading and running the printer dump. Effective, but not the most convenient procedure.

The new version is all rolled into one program. It relocates the plot to A000, under BASIC, immediately before printing. Now the CPU can bank out BASIC, keep the operating system, and send the plot to the printer. The call to print is:

```
SYS 49194,s,d   where (size) s = 0 small printout
                             s = 1 double-size printout
                   (density) d = 0 light background
                             d = 1 dark background
```

Other Features

Adding these capabilities extended the machine code beyond 4K, so I stashed it in the high end of BASIC storage at 9800. It must be protected by POKE 56,152:CLR. (The BASIC loader does this automatically).

The extra space this created was used to add two more features that you may wish to take advantage of, though they aren't used by The Projector. These are a split-screen capability for interactive plotting, and the ability to load Koala Pad pictures for further editing. Here are the commands:

```
SYS 49200,n   splitscreen, n text lines.
              n = 0      all graphics.
              n = 1-15   number of text lines at bottom.
              n = 255    all text.
SYS 49197     put Koala pic at $E000, after loading.
```

Load either a Koala Pad picture or a plot by The Projector; then, SYS 49200,5 in immediate mode will give you a split screen with enough text lines to see what you're doing. You can then proceed to add text labels or any other editing you wish, then print the result.

The 128

The new Commodore 128 has a number of built-in features that help in converting the program, notably line plotting and other high-res commands. As a result, The Projector is written entirely in BASIC. While this is a marvellous improvement, it is the usual 'good news, bad news' situation; the major benefit is that special machine- code routines are not required. One disadvantage is that the ROMs cannot be modified as the 64 routines were, so dealing with hidden lines requires some gyrations in BASIC.

The method used to delete hidden lines is completely different from that used for the 64. This time, plotting proceeds from background to foreground; the PAINT command is used to mask out hidden detail by painting the area underneath each line in background colour. That's a little trickier than it sounds, and requires making five passes at each line. As a result, the plotting is considerably slower than with machine language – the same old slow BASIC problem.

There is one compensating factor with the 128 – the calculations are done in FAST mode at 2 MHz, cutting the preparatory time in half. Don't panic when the screen goes blank; the VIC chip can't keep up at that speed, so output is automatically switched to the 80–column screen temporarily. The picture will soon be back.

The 128's coordinate system is upside–down from Gary Kiziak's, with Y measured from the top of the screen. To match this, all heights are inverted.

Other Program Improvements

Seven new formulae have been developed and are included in the programs; some are reproduced here. To make these easier to access, each plot is selected from a menu, using the DEF FN command. The vertical lines in each grid may be plotted or, as an option, left out.

There is also the capability of plotting empirical data. To illustrate this, I've supplied some data on our famous west coast rainfall. Should you wish to enter your own information on budgets, ground contours, or whatever, here is the data format to use. In the first line:

```
7000 DATA TITLE, M, N, SP
```

M+1 is the number of data points in each row. N+1 is the number of rows to plot. SP is a constant affecting the proportion, typically 100 to 160.

```
7010 DATA a,b,c,...
7020 DATA d,e,f,...
7030 DATA g,h,i,...©
```

There must be (M+1)*(N+1) data points, where a,b,c is the first row, etc.

When plotted in this way, patterns in the data become easy to spot. Note how Vancouver's rainfall increases in the late fall. (Sometimes it seems like it never quits!).

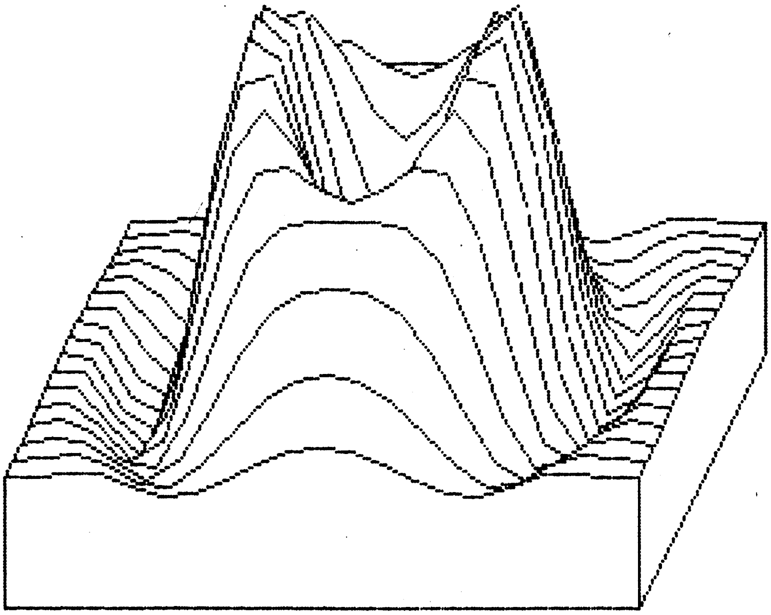
I hope you find that The Projector Part II expands the usefulness of your computer, whether it's a 64 or 128. If you come up with a formula or set of data that produces a particularly interesting plot, I'd be very curious to see it. Perhaps we'll be able to publish a selection of the best plots. Just send a copy to the address below.

Ian Adam P. Eng.
4425 West 12th Avenue
Vancouver BC
V6R 2R3 Canada

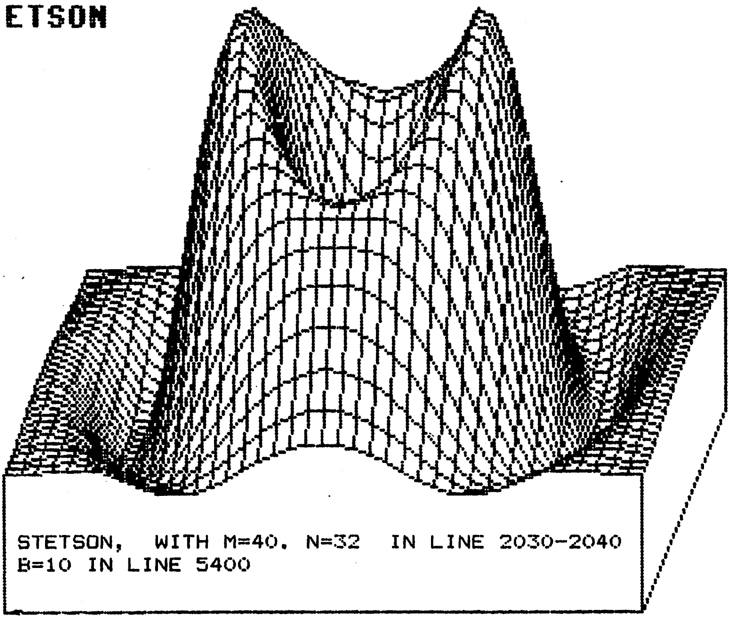
Listing 1: BASIC portion of the new C64 projector program. For this program to work, the files "hiprnt1.ml" and "hiprnt2.ml" must be present on disk when the program is run. These files can be found on the Transactor disk for this issue, or created by the BASIC programs in Listing 3 and Listing 4.

```
NF  1000 printchr$(147)" 64 projector
AD  1010 print" perspective plotter
OG  1020 print" with hidden lines
ME  1030 print" by ian adam
CM  1040 print" vancouver bc
OF  1050 print" december 1985
IJ  1060 :
CJ  1070 rem requires hires plotting routines
IE  1080 rem the transactor vol 5 issue 6
AK  1090 rem with extensions by ia
AM  1100 :
JM  1110 if peek(38912)=1 then a=2
JC  1120 poke 53281,2-a
KJ  1130 on a goto 1150,1190
DM  1140 a=1: load"hiprnt1.ml",8,1
KI  1150 poke 56,152 : clr
DE  1160 load"hiprnt2.ml",8,1
GA  1170 :
AE  1180 rem start here!
KH  1190 gosub1990, constants
BM  1200 gosub2650, choose
KC  1210 gosub2090, config'n
MC  1220 gosub2170, viewing angle
OG  1230 gosub2390, get data
HH  1240 gosub1390, scale
FK  1250 gosub1580, plot
GH  1260 gosub2550, message
KG  1270 :
GH  1280 poke198,0:wait198,1:getb$
PF  1290 if b$="r" then gosub2170:goto1240
IM  1300 if b$="p" then sys hi,0: sys du,0,0: sys te
         : goto1260: dump to printer
OG  1310 if b$="a" then 1250
AF  1320 if b$="n" then if dd then run
HI  1330 if b$="n" then gosub2650:goto1220
MK  1340 if b$="v" then v=1-v
EF  1350 if b$="h" then h=1-h
MN  1360 if b$<>"q" then 1260
KF  1370 end
IN  1380 :
EJ  1390 rem vertical scaling
PA  1400 print:print"scaling data. . .
OO  1410 vscalar=9e9
DM  1420 for y=0 to n
KL  1430 a=z(0,y):for x=1 to m
GI  1440 if z(x,y)>a then a=z(x,y)
GE  1450 next:rem find highest point on line
GF  1460 if a then tmp=(199-yv(y))/a : if vs>tm then vs=tm
EH  1470 next:rem select best feasible scale
MD  1480 :
MH  1490 rem calculate rise
MB  1500 print". . .still scaling!
NB  1510 for y=0 to n
BF  1520 tm=yv(y)
LC  1530 for x=0 to m
BO  1540 r(x,y)=z(x,y)*vs+tm
```

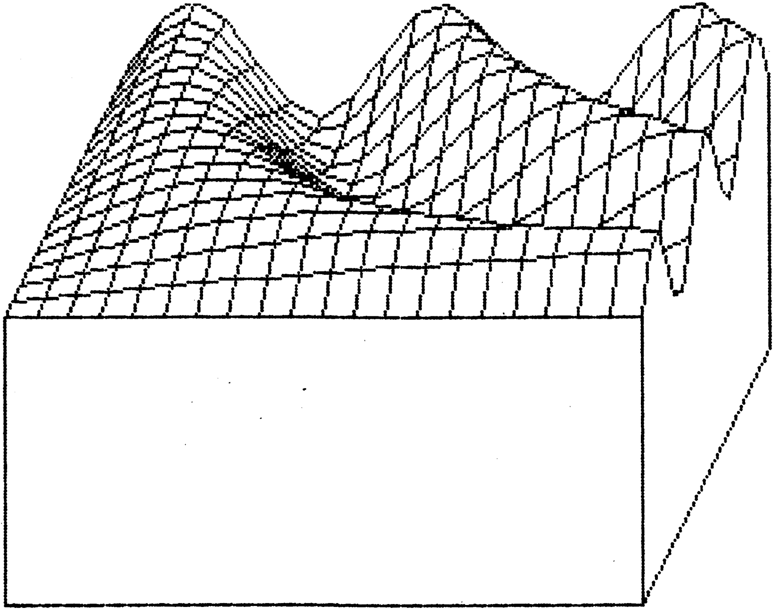
```
DC  1550 nextx,y
ED  1560 return
GJ  1570 :
JK  1580 rem set up screen
FG  1590 syshi,0,0,13
KA  1600 sysdm,1
OL  1610 :
DO  1620 rem plot horizontal lines
PO  1630 sysmo,10,r(0,0)
BE  1640 d1=dr:if h then d1=hd
JK  1650 for y=0 to n
BM  1660 tm=yh(y)
IL  1670 for x=1 to m
FP  1680 sysd1,tm+xh(x),r(x,y)
GK  1690 nextx
MC  1700 if y=n then 1800
CC  1710 :
GJ  1720 rem plot vertical lines
KL  1730 sysdr,yh(y+1)+xh(m),r(m,y+1)
PO  1740 sysd1,yh(y)+xh(m),r(m,y)
FL  1750 for x=m-1 to 0 step-1
AO  1760 if v then x=0
CO  1770 sysmo,tm+xh(x),r(x,y)
JI  1780 sysd1,yh(y+1)+xh(x),r(x,y+1)
DB  1790 next x,y
MH  1800 :
NI  1810 rem draw box
NK  1820 sysmo,10,r(0,0)
FN  1830 sysdr,10,10
LO  1840 sysdr,xh(m),10
IH  1850 sysdr,xh(m),r(m,0)
OA  1860 sysmo,xh(m),10
JD  1870 sysdr,xh(m)+yh(n),yv(n)
CO  1880 sysdr,xh(m)+yh(n),r(m,n)
GN  1890 :
FG  1900 rem title
KB  1910 sysco,8:syspr,1,24,a$
EP  1920 :
CK  1930 rem wait for human
AF  1940 wait198,3:poke198,0
NC  1950 syste:print chr$(147)
EM  1960 return
GC  1970 :
CM  1980 rem  constants
NC  1990 hi=49152:dr=49155:mo=49161
PF  2000 dm=49167:co=49173:te=49179
JD  2010 pr=49182:hd=49191:du=49194
KL  2020 m=20:rem x-dimension
MM  2030 n=16:rem y-dimension
NH  2040 sp=96:rem vertical separation
HF  2050 th=-1
DA  2060 ms$(0)="hide":ms$(1)="show
CD  2070 return
EJ  2080 :
GC  2090 input"hidden lines to be shown (y/n)";b$
KF  2100 h=abs(b$="n")
BM  2110 input"vertical lines to be shown (y/n)";b$
KI  2120 v=abs(b$="n")
OP  2130 dim z(m,n),r(m,n)
JD  2140 dim xh(m),yh(n),yv(n)
CI  2150 return
EO  2160 :
```



STETSON

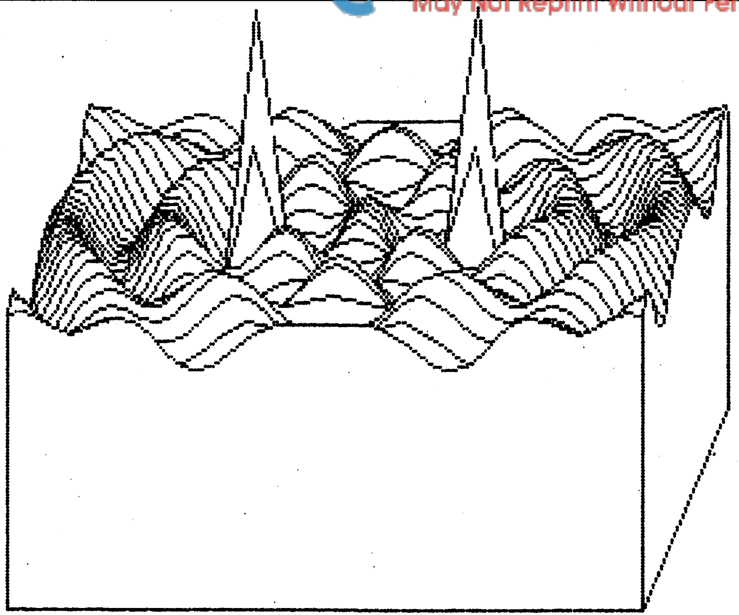


STETSON, WITH M=40. N=32 IN LINE 2030-2040
B=10 IN LINE 5400

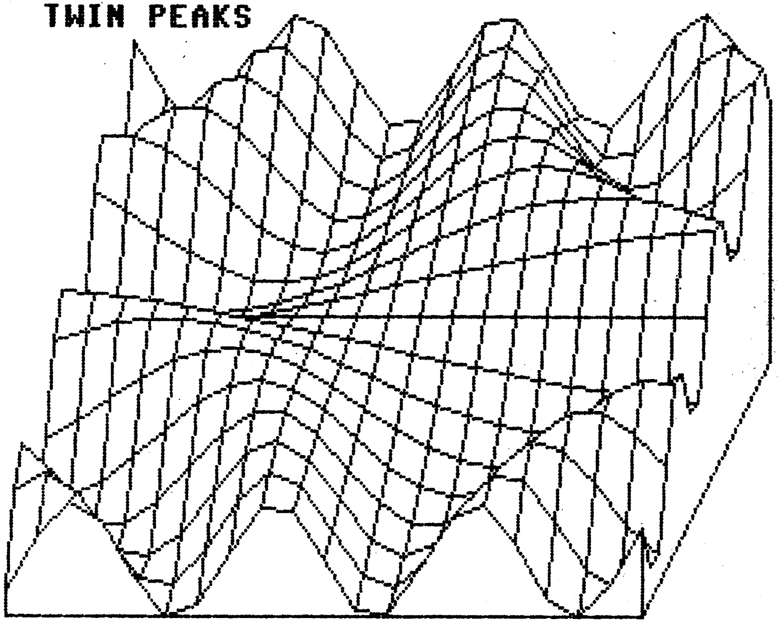


INVERSE WAVES

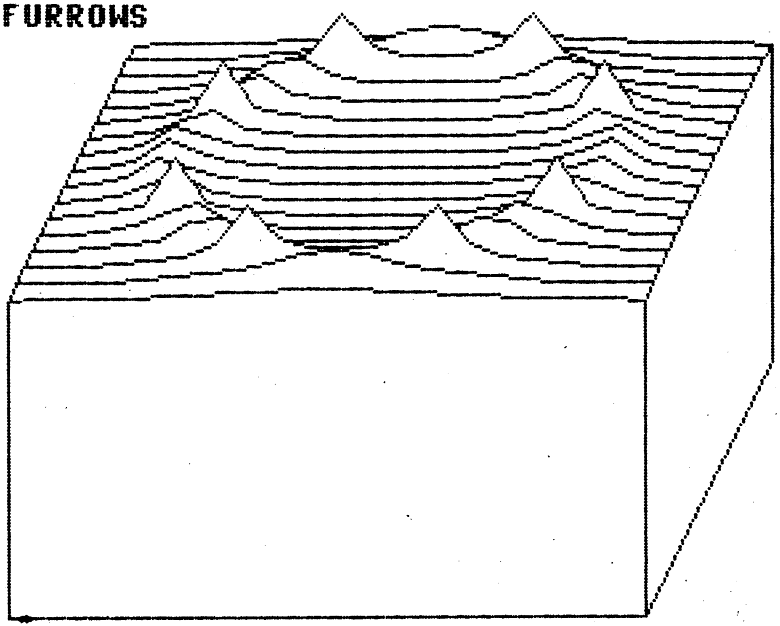
```
KF  2170 rem view angle
JC  2180 if theta<0 then theta=60:rem default angle
NF  2190 print:print"enter viewing angle, or press return
CP  2200 print"for"th"degrees:
KG  2210 inputth :if th<0 or th>90 then 2180
LD  2220 an=th*<clr>/180
DD  2230 tmp=120*cos(an)
GO  2240 xgrid=int((309-tm)/m)
NN  2250 ygrid=int(sp*sin(an)/n)
IJ  2260 ystp=int(tm/n)
CF  2270 :
BK  2280 rem calculate offsets
DC  2290 for x=0 to m
GI  2300 xhriz(x)=10+x*xg
KA  2310 next
HE  2320 for y=0 to n
MA  2330 yhriz(y)=y*ys
NK  2340 yvert(y)=10+y*yg
CD  2350 next
EF  2360 return
GL  2370 :
NG  2380 rem data to plot
JA  2390 print:print"creating data. . .
JA  2400 if dd then 2480
LJ  2410 for x=0 to m
LK  2420 for y=0 to n
OG  2430 if e then r=fnr(x):s=fns(y)
PI  2440 z(x,y)=fnz(x)
DI  2450 nexty:printx;:nextx:return
AB  2460 :
EE  2470 rem  read empirical results from data
HO  2480 for y=0 to n
LO  2490 for x=0 to m
PN  2500 read z(x,y)
KN  2510 nextx:printy;:nexty
EP  2520 return
GF  2530 :
KM  2540 rem *** menus: ***
BB  2550 print chr$(19)chr$(18);" press:": print
GG  2560 print"r review from another angle
LK  2570 print"p send projection to printer
FO  2580 print"h: "ms$(1-h)" hidden lines
KL  2590 print"v: "ms$(1-v)" vertical lines
NO  2600 print"a plot again
HN  2610 print"n for a new shape
OD  2620 print"q quit
CG  2630 return
EM  2640 : .
OB  2650 print:print chr$(18);" press:": print
DM  2660 print"1. stetson
IA  2670 print"2. inverse waves
OB  2680 print"3. furrows
CF  2690 print"4. cascade
JK  2700 print"5. twin peaks
II  2710 print"6. crater
IG  2720 print"7. radial
HO  2730 print"8. read data
IC  2740 :
DP  2750 wait198,1:geta$
DO  2760 e=0:a=val(a$):if a<1 or a>8 then run
LJ  2770 on a gosub 2890,2940,2980,3020,3070,
         3130,3180,2800
```



TWIN PEAKS

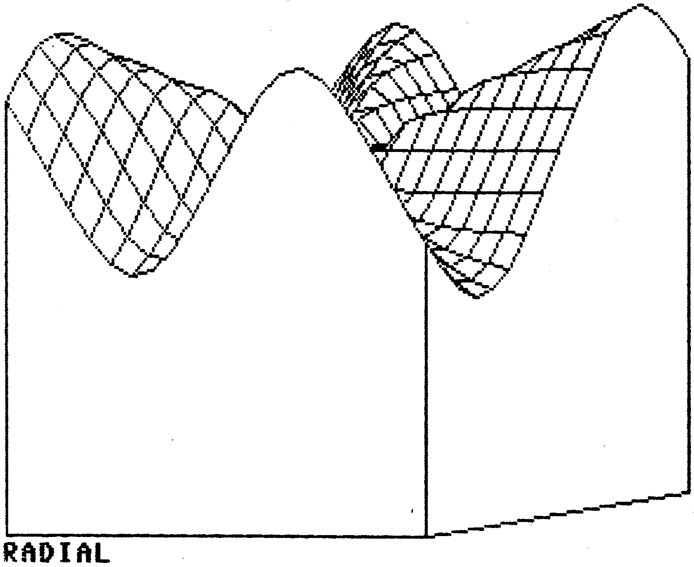


FURROWS

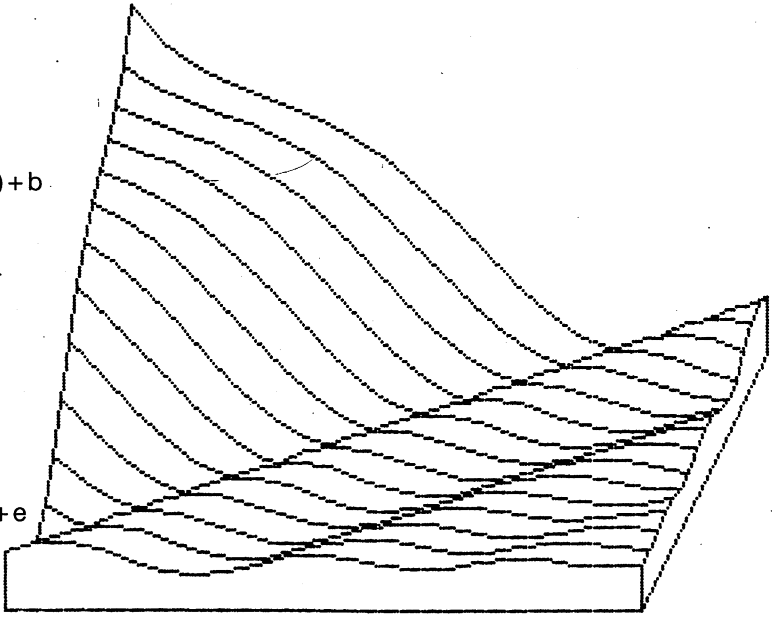


CRATER

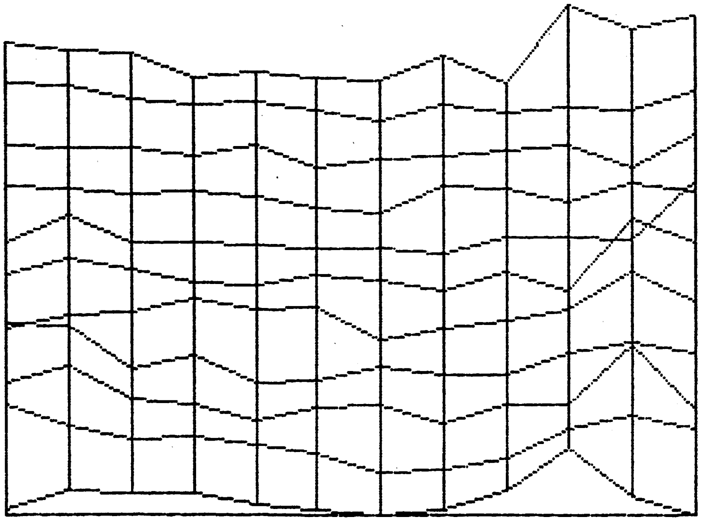
```
HL  2780 printa$:return
KF  2790 :
EL  2800 print:print chr$(18);" press:": print
IC  2810 print"1. rainfall
CI  2820 print"2. more data
BM  2830 print:print"0. first menu
PA  2840 wait198,1:geta:if a=0 or a>2 then run
FG  2850 on a gosub 3230,3470
BN  2860 read a$,m,n,sp
BK  2870 dd=1:return
EL  2880 :
ED  2890 a=m/2:b=5:c=n/2:d=2:e=.2
HA  2900 deffnr(x)=(x-a)/b:deffns(y)=(y-c)/b
JC  2910 deffnz(x)=sin(r*r*d+s*s)*exp(-r*r-s*s)+e
FF  2920 a$="stetson":return
GO  2930 :
IO  2940 a=5
FF  2950 deffnz(x)=sin(x*y/m)+a
OJ  2960 a$="inverse waves":return
OA  2970 :
GJ  2980 a=m/2:b=n/2:c=4:d=1
MP  2990 deffnz(x)=sin((x-a)*(y-b)/b)+y/c+d
EO  3000 a$="furrows":return
GD  3010 :
BA  3020 a=6:b=2:c=.1:e=-1.2
OO  3030 deffnr(x)=y/n-x/m:deffns(y)=r+r
LJ  3040 deffnz(x)=(c+exp(s+r))*cos(a*r*r-a*s+e)+b
MD  3050 a$="cascade":return
IG  3060 :
FD  3070 a=int(m/3):b=m-a:c=n/2:d=3:e=.1:f=.4
MP  3080 deffnr(x)=(x-a)*(x-a)+(y-c)*(y-c)
CB  3090 deffns(y)=(x-b)*(x-b)+(y-c)*(y-c)
KM  3100 deffnz(x)=cos(sqr(r))*(exp(-r/d)+e)
          +cos(sqr(s))*(exp(-s/d)+e)+f
NF  3110 a$="twin peaks":return
EK  3120 :
FA  3130 a=m/2:b=n/2:c=45:e=5
PM  3140 deffnr(x)=abs((x-a)*(x-a)+(y-b)*(y-b)-c)+e
          :deffns(y)=.
OG  3150 deffnz(x)=e/r+e
CG  3160 a$="crater":return
GN  3170 :
JL  3180 a=m/2:b=n/2:c=.001:d=40
HG  3190 deffnz(x)=(abs(x-a)+abs(y-b))*sin(4
          *atn((y-b)/(x-a+c)))+d
MF  3200 a$="radial":return
OP  3210 :
IA  3220 :
EE  3230 poke 65,peek(61): poke 66,peek(62): rem set
          data ptr
EM  3240 return
GC  3250 :
DG  3260 data rainfall in mm   vancouver   1975-1985,
          11,10,160
KD  3270 :
BP  3280 data  30,  94,  83,  90,  44,  31,    7,  29
KK  3290 data  95, 266,    0,    0: rem 1985
CB  3300 data 268, 176, 132, 140, 109,  80,    1,  17
MJ  3310 data  60, 167, 225, 170, 186, 239, 122,  98
HG  3320 data  40,  84, 102,  30,  99,  97, 325,  77
HE  3330 data 247, 229,  68, 116,  18,  28,  74,  44
CE  3340 data  46, 131, 173, 131,  57, 106, 124, 173
```



RADIAL



CASCADE



RAINFALL IN MM   VANCOUVER   1975-1985

```
GL  3350 data 130, 138,  17,  59,  89, 125, 274, 157
MD  3360 data  96, 165, 120,  71,  54, 100,  74,  35
ON  3370 data 104,  40, 319, 218: rem 1980
AP  3380 data  57, 162,  61,  57,  49,  33,  32,  19
PL  3390 data  74,  76,  65, 294, 113,  95,  77,  84
DP  3400 data  65,  23,   9, 104,  96,  42, 124,  88
JB  3410 data 102,  87,  84,  52,  98,  18,  51,  53
JI  3420 data  82,  98,  20, 140, 167, 159, 112,  87
PC  3430 data  95,  67,  24,  84,  53,  81,  64, 135
KH  3440 data 162, 126, 118,  30,  49,  31,  19, 106
FE  3450 data   1, 300, 210, 268: rem 1975
IP  3460 :
ED  3470 poke 65,peek(61): poke 66,peek(62): rem set
         data ptr
EL  3480 return
GB  3490 :
KI  3500 data none entered,1,1,100
```

Listing 2: The version of the new projector for the C128. No additional files are required to run this program. Make sure you switch your monitor to the 40 column (C64) side to see the plots.

```
NI  1000 print chr$(147)"128 projector"
EH  1010 print "perspective plotter"
EN  1020 print "with hidden lines"
OG  1030 print "by  ian  adam"
IC  1040 print "vancouver  bc"
GO  1050 print "december 1985"
IJ  1060 :
NM  1070 trap 2170
DO  1080 gosub 2010, constants
IF  1090 gosub 2760, choose
JP  1100 gosub 2090, config
OL  1110 gosub 2230, viewing angle
AA  1120 gosub 2510, get data
OJ  1130 gosub 1270, scale
BD  1140 gosub 1460, plot
EF  1150 do
IB  1160 gosub 2670, message
LO  1170 getkey b$
MO  1180 if b$="r" then gosub 2230: goto 1130
MB  1190 if b$="p" then 1140
IN  1200 if b$="n" then if dd then run
PA  1210 if b$="n" then gosub 2760: goto 1110
ED  1220 if b$="v" then v=1-v
MN  1230 if b$="h" then h=1-h
MF  1240 if b$="q" then end
GF  1250 :
EM  1260 loop
MB  1270 rem vertical scaling
MG  1280 vscalar=9e9
BE  1290 for y=0 to n
II  1300 :
CE  1310 a=z(0,y):for x=1 to m
OA  1320 if z(x,y)>a then a=z(x,y)
GD  1330 next
CG  1340 if a then if vs>yv(y)/a then vs=yv(y)/a
MP  1350 next:rem select best feasible scale
BK  1360 color4,7
OM  1370 :
OA  1380 rem calculate rise
FK  1390 for y=0 to n
JN  1400 tm=yv(y)
DL  1410 for x=0 to m
PA  1420 r(x,y)=tm-z(x,y)*vs
LK  1430 next x,y
ML  1440 return
OB  1450 :
BD  1460 rem set up screen
KK  1470 color 1,14: graphic 1,1
GD  1480 slow
GE  1490 :
LG  1500 rem plot horizontal lines
II  1510 locate g(0,n),r(0,n)
MJ  1520 for y=n to 0 step-1
MH  1530 if y=n then 1620
IH  1540 :
MO  1550 rem plot vertical lines
HP  1560 for x=m-1 to 0 step-1
CC  1570 if v then x=0
GG  1580 locate g(x,y+1),r(x,y+1)
OD  1590 draw to g(x,y),r(x,y)
ME  1600 next x
LJ  1610 if h then gosub 1850,mask
GI  1620 for x=1 to m
GG  1630 draw to g(x,y),r(x,y)
NH  1640 next x,y
GO  1650 :
HP  1660 rem draw box
EL  1670 locate 10,r(0,0)
KL  1680 draw to 10,ht
NH  1690 draw to xm,ht
HB  1700 draw to g(m,n),yv(n)
IE  1710 draw to g(m,n),r(m,n)
LJ  1720 for y=n-1 to 0 step-1
JK  1730 draw to g(m,y),r(m,y)
AN  1740 next
JL  1750 draw to xm,ht
EF  1760 :
DO  1770 rem title
AA  1780 char 1,1,24,a$
CH  1790 :
AC  1800 rem wait for human
LG  1810 getkey b$
JE  1820 graphic 0,1
CE  1830 return
EK  1840 :
KC  1850 rem mask hidden lines
GK  1860 for i=-1 to 1
EF  1870 locate g(0,y)+i,r(0,y)+3
KI  1880 for x=1 to m
LM  1890 draw to g(x,y)+i,r(x,y)+3
BH  1900 next x,i
IC  1910 locate g(0,y),r(0,y)+1
CL  1920 for x=1 to m
AP  1930 draw 0,+0,+0 to g(x,y),r(x,y)+1
AK  1940 next x
MF  1950 draw 0,+0,+0 to +8,+8
LA  1960 paint 0,g(m,y),r(m,y)+3
EI  1970 paint 0,g(0,y),r(0,y)+3
FJ  1980 locate g(0,y),r(0,y)
CO  1990 return
EE  2000 :
```

```
AL  2010 m=20: rem x-dimension
CM  2020 n=16: rem y-dimension
DH  2030 sp=96: rem vertical separation
PO  2040 ms$(0)="hide": ms$(1)="show
IE  2050 ht=190: th=-1
PO  2060 pi=3.14159265
CD  2070 return
EJ  2080 :
HE  2090 print "hidden lines to be shown (y/n)": getkey b$
HE  2100 h=abs(b$<>"y")
EE  2110 print "vertical lines to be shown (y/n)": getkey b$
HH  2120 v=abs(b$<>"y")
MH  2130 dim z(m,n),r(m,n),g(m,n)
LF  2140 dim xh(m),yv(n)
CI  2150 return
EO  2160 :
JK  2170 rem error trap
CP  2180 slow
NJ  2190 print err$(er)el
EO  2200 graphic 0
CK  2210 end
AC  2220 :
GJ  2230 rem view angle
AE  2240 print"** screen will be blanked a while **"
PG  2250 if theta<0 then theta=60: rem default angle
FM  2260 print: print"enter viewing angle, or press return"
OJ  2270 print"for"th"degrees:"
OA  2280 input th
EJ  2290 an=th*pi/180
HG  2300 fast
IC  2310 tmp=120*abs(cos(an))
GD  2320 xgrid=int((309-tm)/m)
FL  2330 ygrid=int(sp*abs(sin(an))/n)
IO  2340 ystp=int(tm/n)
CK  2350 :
BP  2360 rem calculate offsets
DH  2370 for x=0 to m
GN  2380 xhriz(x)=10+x*xg
KF  2390 next
MH  2400 xm=xh(m)
JL  2410 color 4,5
LK  2420 for y=0 to n
NN  2430 yvert(y)=ht-y*yg
HO  2440 tm=y*ys
DM  2450 for x=0 to m
AJ  2460 g(x,y)=xh(x)+tm
LL  2470 next x,y
MM  2480 return
OC  2490 :
FO  2500 rem data to plot
NF  2510 if dd then 2600
PA  2520 for y=0 to n
DB  2530 for x=0 to m
MN  2540 if e then r=fnr(x):s=fns(y)
NP  2550 z(x,y)=fnz(x)
BL  2560 next x: color 4,(yand15)+1: next y
GC  2570 return
II  2580 :
DB  2590 rem read empirical results
PF  2600 for y=0 to n
DG  2610 for x=0 to m
HF  2620 read z(x,y)
```

```
HP  2630 next x: color 4,(yand15)+1: next y
MG  2640 return
OM  2650 :
CO  2660 rem ** menus **
HH  2670 print chr$(19)" press:"
EE  2680 print "r review from another angle"
JL  2690 print "h: "ms$(1-h)" hidden lines"
KE  2700 print "v: "ms$(1-v)" vertical lines"
LN  2710 print "p plot again"
HG  2720 print "n for a new shape"
OM  2730 print "q quit"
AN  2740 return
CD  2750 :
HA  2760 color 0,1
JE  2770 print " press:"
NF  2780 print "1. stetson"
EM  2790 print "2. inverse waves"
IL  2800 print "3. furrows"
MO  2810 print "4. cascade"
HI  2820 print "5. twin peaks"
II  2830 print "6. crater"
IG  2840 print "7. radial"
DK  2850 print "8. read data"
AK  2860 :
LI  2870 getkey a$
AE  2880 e=0: a=val(a$)
LC  2890 if a<1 or a>8 then run
HO  2900 on a gosub 3030,3080,3120,3160,3210,
             3270,3320,2930
JD  2910 print a$: return
MN  2920 :
NI  2930 print " press"
OO  2940 print "1. rainfall"
IE  2950 print "2. more data"
JB  2960 print "0. first menu"
LN  2970 getkey a: if a=0 or a>2 then run
DG  2980 if a=1 then restore 3360
AH  2990 if a=2 then restore 3560
NF  3000 read a$,m,n,sp
NC  3010 dd=1: return
AE  3020 :
AM  3030 a=m/2: b=5: c=n/2: d=2: e=.2
DJ  3040 deffnr(x)=(x-a)/b:deffns(y)=(y-c)/b
FL  3050 deffnz(x)=sin(r*r*d+s*s)*exp(-r*r-s*s)+e
BO  3060 a$="stetson": return
CH  3070 :
EH  3080 a=5
BO  3090 deffnz(x)=sin(x*y/m)+a
KC  3100 a$="inverse waves": return
KJ  3110 :
CC  3120 a=m/2:b=n/2:c=4:d=1
II  3130 deffnz(x)=sin((x-a)*(y-b)/b)+y/c+d
AH  3140 a$="furrows": return
CM  3150 :
NI  3160 a=6: b=2: c=.1: e=-1.2
KH  3170 deffnr(x)=y/n-x/m:deffns(y)=r+r
HC  3180 deffnz(x)=(c+exp(s+r))*cos(a*r*r-a*s+e)+b
IM  3190 a$="cascade": return
EP  3200 :
BM  3210 a=int(m/3): b=m-a: c=n/2: d=3: e=.1: f=.4
II  3220 deffnr(x)=(x-a)*(x-a)+(y-c)*(y-c)
OJ  3230 deffns(y)=(x-b)*(x-b)+(y-c)*(y-c)
```

```
GF  3240 deffnz(x)=cos(sqr(r))*(exp(-r/d)+e)
          +cos(sqr(s))*(exp(-s/d)+e)+f
JO  3250 a$="twin peaks": return
AD  3260 :
BJ  3270 a=m/2: b=n/2: c=45: e=5
LF  3280 deffnr(x)=abs((x-a)*(x-a)+(y-b)*(y-b)-c)+e
          :deffns(y)=.
KP  3290 deffnz(x)=e/r+e
OO  3300 a$="crater": return
CG  3310 :
FE  3320 a=m/2: b=n/2: c=.001: d=40
DP  3330 deffnz(x)=(abs(x-a)+abs(y-b))*sin(4
          *atn((y-b)/(x-a+c)))+d
IO  3340 a$="radial": return
KI  3350 :
HM  3360 data rainfall in mm   vancouver   1975-1985,
          11, 10, 160
LE  3370 data  30,  94,  83,  90,  44,  31,   7,  29
EA  3380 data  95, 266,   0,   0: rem 1985
MG  3390 data 268, 176, 132, 140, 109,  80,   1,  17
GP  3400 data  60, 167, 225, 170, 186, 239, 122,  98
BM  3410 data  40,  84, 102,  30,  99,  97, 325,  77
BK  3420 data 247, 229,  68, 116,  18,  28,  74,  44
MJ  3430 data  46, 131, 173, 131,  57, 106, 124, 173
AB  3440 data 130, 138,  17,  59,  89, 125, 274, 157
GJ  3450 data  96, 165, 120,  71,  54, 100,  74,  35
ID  3460 data 104,  40, 319, 218: rem 1980
KE  3470 data  57, 162,  61,  57,  49,  33,  32,  19
JB  3480 data  74,  76,  65, 294, 113,  95,  77,  84
NE  3490 data  65,  23,   9, 104,  96,  42, 124,  88
DH  3500 data 102,  87,  84,  52,  98,  18,  51,  53
DO  3510 data  82,  98,  20, 140, 167, 159, 112,  87
JI  3520 data  95,  67,  24,  84,  53,  81,  64, 135
EN  3530 data 162, 126, 118,  30,  49,  31,  19, 106
PJ  3540 data   1, 300, 210, 268: rem 1975
CF  3550 :
MM  3560 data none entered,,,100
```

Listing 3: Run this program to create the file "hiprnt1.ml" for the C64 projector in listing 1.

```
AI  1000 rem* program to create 'hiprnt1.ml' on disk *
HD  1010 for j=1 to 2640 : read x
BJ  1020 ch=ch+x : next
AM  1030 if ch<>325033 then print"checksum error" : end
FM  1040 print "data ok, now creating file": print
CD  1050 restore
MF  1060 open8,8,8,"0:hiprnt1.ml,p,w"
EM  1070 print#8,chr$(0)chr$(192);
NH  1080 for j=1 to 2640 : read x
CM  1090 print#8,chr$(x); : next
ED  1100 close 8
OF  1110 print "prg file 'hiprnt1.ml' created. . .
KF  1120 print "this generator no longer needed.
II  1130 rem
JO  1140 data  76, 222, 195,  76,  32, 198,  76, 118, 197
DA  1150 data  76, 124, 196,  76,  44, 196,  76, 231, 198
IC  1160 data  76, 245, 198,  76,  28, 199,  76,  84, 199
CE  1170 data  76, 202, 195,  76, 223, 199,  76, 228, 201
GO  1180 data  76,  33, 202,  76,  27, 198,  76,  63, 152
CP  1190 data  76, 236, 155,  76, 148, 195,   0,   0,   0
BK  1200 data   0, 255, 128,   0,   7, 248,   0, 129,   0
AD  1210 data   0,   0,   0,   1,   0,  15, 240, 240,   0
NJ  1220 data   0, 208,   0,   0,   0,   0, 219, 219, 255
HJ  1230 data   0,   0,   0,   0,   0,   0,   0, 255, 129
OF  1240 data 223,   0, 219,   0, 239,  64, 223,   9, 255
PG  1250 data 194, 255,  50, 255,   0, 255, 202, 155, 139
GO  1260 data 239,   9, 219, 219, 255,   0, 223,   0, 222
EC  1270 data   1, 255,  25, 255,   8, 206, 194, 219, 219
NM  1280 data 207, 211,   0, 254,   0, 246,   0, 255,   0
CH  1290 data 255,   0, 125,  36, 255,   0, 255,   0, 166
OA  1300 data   0, 125,   0, 254,  20, 255,   0, 247,  17
FB  1310 data 255,   0, 125,  64, 254,   0, 125,  32, 255
DP  1320 data   0, 255,   0, 239,   0, 255,   4, 255,  36
JA  1330 data 100,   0, 255,   0, 246,   0, 254,   0, 255
CB  1340 data   0, 255,   0, 255,   0, 255,   0, 255,  48
LD  1350 data 254,   0, 255,   0, 255,   0, 254,   0, 255
LP  1360 data   0, 255,  32,  53,  36, 255, 117, 255,   0
AE  1370 data 109,   0, 255,   0, 255,   0, 255,   0, 255
PI  1380 data   0, 126,  32, 255,  77, 255,  16, 255,  32
BC  1390 data 215,   0, 121,   0, 207,   0, 255,   0, 117
GL  1400 data 108, 117,  64, 247, 109,  36,   0, 255,   0
LC  1410 data 255,  32, 255,   0, 254,   0, 247,  49,  61
EI  1420 data 100, 108,  48,  44, 255,   1, 255,  25, 255
PK  1430 data  64, 255,   0, 255, 130,  91,   0, 255,   0
OP  1440 data 255,  89, 255, 130, 255,  17, 255,   0, 127
BL  1450 data  12, 238,   0, 255, 202, 189,   1, 255, 194
KC  1460 data 223,   0, 255,   0, 255,  16, 255,   0, 251
IP  1470 data   0, 219, 155, 255,   8, 255,  25, 255,  48
AM  1480 data 255,   0, 127,   0, 255,   0, 255,   0, 255
AJ  1490 data   0, 223,   1, 255,   0, 255,   0, 255,  17
NL  1500 data 255,   0, 255,   0, 223, 202, 219,   9, 200
PM  1510 data   0, 219, 219, 255,   0, 255,   0, 255,   0
MN  1520 data 255,   0, 255, 129, 223,   0, 218,   0, 239
GC  1530 data  64, 223,   8, 255, 194, 255,  50, 255,   0
KC  1540 data 255, 202, 155, 139, 255,   8, 218, 219, 255
ML  1550 data   0, 223,   0, 222,   1, 255,  25, 255,   8
KP  1560 data 206, 194, 219, 219,  95, 211,   0, 254,   0
OA  1570 data 246,   0, 255,   0, 255,   0, 125, 164, 255
GN  1580 data   0, 255,   0, 166,   0, 125,   0, 254,  20
KE  1590 data 255,   0, 247,  17, 255,   0, 125,  66, 254
FP  1600 data   0, 253,  32, 255,   0, 255,   0, 239,   0
PI  1610 data 255,   0, 251,  36, 100,   0, 255,   0, 230
JA  1620 data   0, 231,   0, 255,   0, 255,   0, 173,  43
IK  1630 data 195, 208,  87, 173,   0, 221, 141,  45, 195
OP  1640 data 173,  24, 208, 141,  43, 195, 173,  17, 208
FN  1650 data  41, 127, 141,  17, 208, 141,  41, 195, 173
GI  1660 data  22, 208, 141,  56, 195, 173,   0,   3, 201
KP  1670 data  63, 208,   7, 173,   1,   3, 201, 194, 240
NL  1680 data  44, 173,   0,   3, 141,  66, 194, 173,   1
II  1690 data   3, 141,  67, 194, 169,  63, 141,   0,   3
CM  1700 data 169, 194, 141,   1,   3, 173,   2,   3, 141
PO  1710 data 134, 194, 173,   3,   3, 141, 135, 194, 169
MP  1720 data  96, 141,   2,   3, 169, 194, 141,   3,   3
PF  1730 data  96, 169, 127, 141,  13, 220, 173,  61, 192
JJ  1740 data  48,   6,  32,  88, 195,  32,  13, 196, 165
JB  1750 data   1, 141,  60, 192,  41, 252, 133,   1,  96
DC  1760 data 173,  60, 192, 133,   1, 173,  61, 192, 141
GK  1770 data  13, 220,  48,   3,  32, 173, 195,  96,  16
LO  1780 data   3,  76, 139, 227, 142,  13,   3,  44,  74
BF  1790 data 192,  16, 245, 169,   0, 133,  20, 169,   0
OH  1800 data 133,  21, 162, 250, 154, 169, 167,  72, 169
GC  1810 data 233,  72,  76, 163, 168,  44,  61, 192,  16
```

```
LB  1820 data   3,  32, 202, 195, 173,  66, 194, 141,   0
KG  1830 data   3, 173,  67, 194, 141,   1,   3, 173, 134
EG  1840 data 194, 141,   2,   3, 173, 135, 194, 141,   3
KG  1850 data   3, 169,   0, 141,  74, 192,  76, 131, 164
FK  1860 data 164, 254, 240,  13, 160,   0, 145, 251, 200
OH  1870 data 208, 251, 230, 252, 198, 254, 208, 243, 164
FM  1880 data 253, 240,  10, 136, 240,   5, 145, 251, 136
FO  1890 data 208, 251, 145, 251,  96, 160,   0, 132, 251
MH  1900 data 160, 204, 132, 252, 160, 232, 132, 253, 160
FD  1910 data   3, 132, 254,  32, 136, 194, 169,   0, 168
AI  1920 data 153, 119, 192, 136, 208, 250, 160,  63, 153
FJ  1930 data 119, 193, 136,  16, 250, 133, 251, 162, 224
CN  1940 data 134, 252, 162,  64, 134, 253, 162,  31, 134
JE  1950 data 254,  76, 136, 194,  32, 253, 174,  32, 138
KA  1960 data 173,  32, 247, 183, 166,  21, 165,  20,  96
PB  1970 data  32, 253, 174,  32, 224, 194, 141,  83, 192
LC  1980 data 142,  84, 192,  32, 221, 194, 141,  85, 192
KI  1990 data 142,  86, 192, 138, 208,  21, 169,  63, 162
JF  2000 data   1,  44,  57, 192,  16,   3, 169, 159, 202
JD  2010 data 205,  83, 192, 138, 237,  84, 192, 176,   3
AI  2020 data  76,  72, 178, 169, 199, 205,  85, 192, 144
OB  2030 data 246,  96, 173,  18, 208, 240,  32, 169,  27
FN  2040 data 162,   0, 160, 199, 141,  17, 208, 142,  24
KB  2050 data 208, 140,   0, 221, 169, 200, 141,  22, 208
HN  2060 data 162,   1, 142,  25, 208, 202, 142,  18, 208
LM  2070 data  76, 188, 254, 169, 218, 141,  18, 208, 169
JK  2080 data   1, 141,  25, 208,  32,  13, 196,  76,  49
CH  2090 data 234, 120, 238,  18, 208, 238,  18, 208, 162
GA  2100 data   0, 142,  26, 208, 173,  25, 208, 208,   3
FN  2110 data 232, 208, 248, 141,  25, 208,  88,  96, 120
OM  2120 data 169,  49, 162, 234, 160, 129, 120, 208,  65
HO  2130 data 173,  41, 195, 141,  17, 208, 173,  43, 195
LH  2140 data 141,  24, 208, 173,  45, 195, 141,   0, 221
NL  2150 data 173,  56, 195, 141,  22, 208,  96,  32, 183
PJ  2160 data 193,  32, 221, 194,  48,  46, 240, 105,  41
MM  2170 data  15,  42,  42,  42, 105,   5,  73, 255, 141
AL  2180 data  73, 195, 141,  18, 208, 120, 169,   1, 141
KB  2190 data  25, 208, 141,  26, 208, 169,  35, 162, 195
GH  2200 data 160, 127, 141,  20,   3, 142,  21,   3, 140
LJ  2210 data  13, 220, 140,  61, 192,  88,  96, 173,  43
KN  2220 data 195, 240,  14,  32,  88, 195,  32, 113, 195
OP  2230 data  32, 123, 195, 169,   0, 141,  43, 195,  96
KN  2240 data  32, 183, 193,  32, 221, 194, 240,   6, 169
CD  2250 data 128, 160,   3, 208,   2, 160,   7, 141,  57
MI  2260 data 192, 140,  58, 192, 152,  73, 255, 141,  59
IB  2270 data 192, 169, 255, 141,  55, 192,  32, 121,   0
PH  2280 data 240,   3,  32,  44, 196,  32,  88, 195,  32
OI  2290 data 113, 195, 173,   0, 221,  41, 252, 141,   0
NI  2300 data 221, 169,  56, 141,  24, 208, 169,  59, 141
DE  2310 data  17, 208, 169, 200,  44,  57, 192,  16,   2
JE  2320 data 169, 216, 141,  22, 208,  96, 169,   1, 141
IM  2330 data  66, 192, 173,  68, 192, 141,  67, 192, 169
BL  2340 data 128, 141,  56, 192,  32, 235, 194, 173,  85
AA  2350 data 192,  10,  10,  10,  10, 141,  63, 192, 141
GD  2360 data  71, 192, 173,  83, 192,  41,  15, 141,  62
IH  2370 data 192,  44,  57, 192,  48,  12,  13,  63, 192
LN  2380 data 141,  63, 192, 141,  71, 192,  76, 168, 194
ME  2390 data 141,  33, 208,  32, 221, 194,  41,  15, 141
PL  2400 data  64, 192,  32, 221, 194, 141,  65, 192, 173
KP  2410 data  63, 192,  76, 168, 194,  32, 235, 194, 162
KF  2420 data   3, 189,  83, 192, 157,  51, 192, 202,  16
OL  2430 data 247,  96,  44,  82, 192,  16,  27, 169, 119
NN  2440 data 133, 251,  24, 169, 192, 109,  52, 192, 133
HP  2450 data 252, 172,  51, 192, 177, 251, 205,  53, 192
EB  2460 data 176, 228, 173,  53, 192, 145, 251,  56, 169
EO  2470 data 199, 237,  53, 192,  72,  74,  74,  74, 133
LI  2480 data 252, 160,   0, 132, 251,  74, 102, 251,  74
MJ  2490 data 102, 251, 101, 252, 133, 252, 173,  51, 192
IP  2500 data 174,  52, 192,  45,  59, 192,  44,  57, 192
LE  2510 data  16,   6,  10,  72, 138,  42, 170, 104,  24
PM  2520 data 101, 251, 133, 251, 138, 101, 252, 133, 252
HN  2530 data 104,  41,   7,  24, 101, 251, 133, 251, 133
LB  2540 data 253, 165, 252,  74, 102, 253,  74, 102, 253
ID  2550 data  74, 102, 253, 133, 254,  44,  57, 192,  48
KD  2560 data   9, 169, 204,   5, 254, 133, 254,  76,  21
FL  2570 data 197, 173,  66, 192, 201,   3, 144, 240, 169
CP  2580 data 216,   5, 254, 133, 254, 165, 252,   9, 224
MK  2590 data 133, 252, 173,  51, 192,  45,  58, 192, 170
OL  2600 data 169,   0, 168,  44,  56, 192,  16,   4, 112
ME  2610 data  20,  80,  15,  36,   2,  48,   9, 169, 255
DJ  2620 data 133,   2,  36, 107,  48,   1,  96, 177, 251
BA  2630 data  77,  55, 192,  44,  57, 192,  48,  10,  61
PG  2640 data 106, 197, 133,  97, 189, 106, 197, 208,   8
CC  2650 data  61, 114, 197, 133,  97, 189, 114, 197,  73
PN  2660 data 255,  49, 251,   5,  97, 145, 251, 177, 253
DK  2670 data  45,  67, 192,  13,  71, 192, 145, 253,  96
HA  2680 data 128,  64,  32,  16,   8,   4,   2,   1, 192
JK  2690 data  48,  12,   3,  32, 124, 196,  32, 121,   0
GD  2700 data 240,  11,  32, 245, 198,  32, 121,   0, 240
BJ  2710 data   3,  32, 231, 198,  32,  20, 194,  32, 171
HP  2720 data 196,  76,  46, 194, 169,   1, 149, 106, 169
FK  2730 data   0, 149, 107,  56, 189,  83, 192, 253,  51
II  2740 data 192, 149,  98, 189,  84, 192, 253,  52, 192
EF  2750 data 149,  99,  16,  20, 169, 255, 149, 106, 149
MD  2760 data 107,  56, 169,   0, 245,  98, 149,  98, 169
OH  2770 data   0, 245,  99, 149,  99,  96,  21,  98, 208
KN  2780 data   4, 149, 106, 149, 107,  96, 165,  99,  74
KC  2790 data 133, 103, 165,  98, 106, 133, 102,  24, 169
IB  2800 data   0, 229,  98, 133, 104, 169,   0, 229,  99
PG  2810 data 133, 105,  36, 107,  16, 228,  32, 127, 196
GP  2820 data  56, 169,   0, 229, 108, 133, 108, 169,   0
LP  2830 data 229, 109, 133, 109,  96,  24, 165, 102, 101
HN  2840 data 100, 133, 102, 170, 165, 103, 101, 101, 133
KG  2850 data 103, 197,  99, 144,  18, 208,   4, 228,  98
AF  2860 data 144,  12, 138, 229,  98, 133, 102, 165, 103
MN  2870 data 229,  99, 133, 103,  56,  96, 169, 128, 141
MB  2880 data  82, 192,  32, 235, 194,  32, 121,   0, 208
CL  2890 data   6,  32,  95, 198,  76,  89, 198, 201, 164
IE  2900 data 208,  16,  32, 127, 196,  32, 115,   0,  32
BD  2910 data 238, 194,  32, 121,   0, 201,  44, 208,  13
LG  2920 data  32, 245, 198,  32, 121,   0, 201,  44, 208
MJ  2930 data   3,  32, 231, 198,  32,  95, 198,  32, 121
MA  2940 data   0, 201, 164, 240, 220, 169,   0, 141,  82
AB  2950 data 192,  96,  32,  20, 194, 162,   0, 134,   2
HA  2960 data  32, 146, 197, 162,   2,  32, 146, 197, 165
KG  2970 data  98, 197, 100, 165,  99, 229, 101, 144,  39
GL  2980 data  32, 202, 197,  32, 139, 196, 230, 104, 208
BJ  2990 data   4, 230, 105, 240,  84, 238,  51, 192, 208
MM  3000 data   3, 238,  52, 192,  32, 246, 197, 144, 232
OA  3010 data  24, 173,  53, 192, 101, 108, 141,  53, 192
GD  3020 data  76, 123, 198, 162,   1, 181,  98, 180, 100
LB  3030 data 149, 100, 148,  98, 181, 106, 180, 108, 149
FO  3040 data 108, 148, 106, 202,  16, 237,  32, 202, 197
DK  3050 data  32, 139, 196, 230, 104, 240,  28, 238,  53
```

```
KL  3060 data 192,  32, 246, 197, 144, 241,  24, 173,  51
IC  3070 data 192, 101, 108, 141,  51, 192, 173,  52, 192
CJ  3080 data 101, 109, 141,  52, 192,  76, 183, 198,  36
ID  3090 data 107,  16,   3,  32, 139, 196,  32, 127, 196
HJ  3100 data  76,  46, 194,  32, 221, 194,  41,   3,  73
HN  3110 data   3, 106, 106, 106, 141,  56, 192,  96,  32
NF  3120 data 221, 194,  41,   3, 240,  27,  44,  57, 192
EE  3130 data  16,  22, 141,  66, 192, 170, 189,  62, 192
FP  3140 data 141,  71, 192, 189,  67, 192, 141,  67, 192
LL  3150 data 189,  24, 199, 141,  55, 192,  96,   0,  85
CC  3160 data 170, 255,  32, 221, 194,  10,  10,  10,  10
BA  3170 data 141,  63, 192,  44,  57, 192,  48,   9,  13
NI  3180 data  62, 192, 141,  63, 192,  76,  68, 199,  32
LG  3190 data 221, 194,  41,  15, 141,  64, 192,  32, 221
BK  3200 data 194,  41,  15, 141,  65, 192, 174,  66, 192
ID  3210 data 189,  62, 192, 141,  71, 192, 189,  67, 192
MM  3220 data 141,  67, 192,  96,  32, 124, 196,  32, 235
HO  3230 data 194, 162,   3, 189,  83, 192, 157,  77, 192
GK  3240 data 202,  16, 247,  32, 121,   0, 240,  11,  32
KL  3250 data 245, 198,  32, 121,   0, 240,   3,  32, 231
EE  3260 data 198,  24, 173,  51, 192, 109,  77, 192, 141
LF  3270 data  83, 192, 173,  52, 192, 109,  78, 192, 141
OF  3280 data  84, 192, 173,  53, 192, 141,  85, 192, 173
BD  3290 data  54, 192, 141,  86, 192,  32,   3, 195,  32
AG  3300 data  95, 198,  56, 173,  85, 192, 237,  79, 192
NJ  3310 data 141,  85, 192, 173,  86, 192, 237,  80, 192
AE  3320 data 141,  86, 192,  32,  27, 195,  32,  95, 198
ID  3330 data  56, 173,  83, 192, 237,  77, 192, 141,  83
CO  3340 data 192, 173,  84, 192, 237,  78, 192, 141,  84
DG  3350 data 192,  32,  95, 198,  24, 173,  85, 192, 109
NM  3360 data  79, 192, 141,  85, 192, 173,  86, 192, 109
BK  3370 data  80, 192, 141,  86, 192,  76,  95, 198, 169
AE  3380 data   0, 133, 251, 133, 252,  32, 241, 183, 224
OK  3390 data  40, 144,   3,  76,  72, 178, 142,  75, 192
CN  3400 data  32, 241, 183, 142,  76, 192, 138, 240,  18
HN  3410 data 224,  25, 176, 237,  24, 165, 251, 105,  40
AC  3420 data 133, 251, 144,   2, 230, 252, 202, 208, 242
FE  3430 data  24, 173,  75, 192, 101, 251, 133, 251, 133
EH  3440 data 253, 133,   3, 169,   0, 101, 252, 133, 252
PD  3450 data  24,  72, 105, 216, 133, 254, 104, 105, 204
EL  3460 data 133,   4,   6, 251,  38, 252,   6, 251,  38
NI  3470 data 252,   6, 251,  38, 252,  24, 165, 252, 105
LJ  3480 data 224, 133, 252, 173,  61, 192,  72,  48,   3
HM  3490 data  32,   7, 196,  32, 253, 174,  32, 158, 173
CK  3500 data  32, 143, 173,  32, 166, 182, 170, 160,   0
EK  3510 data 232, 202, 208,   7, 104,  48,   3,  32, 173
BA  3520 data 195,  96, 177,  34,  32, 105, 200, 200,  76
FG  3530 data  86, 200, 133, 215, 138,  72, 152,  72, 165
OK  3540 data 215,  48,  17, 201,  32, 144,  28, 201,  96
KN  3550 data 144,   4,  41, 223, 208,   2,  41,  63,  76
BI  3560 data 139, 201,  41, 127, 201, 127, 208,   2, 169
AE  3570 data  94, 201,  32, 144, 125,  76, 137, 201, 201
HC  3580 data  14, 208,   6,  32, 248, 201,  76, 189, 201
ON  3590 data 201,  17, 208,  11, 162,  40,  32, 100, 201
NE  3600 data 202, 208, 250,  76, 189, 201, 201,  18, 208
PF  3610 data   8, 169,   1, 141,  81, 192,  76, 189, 201
GE  3620 data 201,  29, 208,   6,  32, 100, 201,  76, 189
CA  3630 data 201, 162,   3,  44, 162,  15, 221, 237, 200
AG  3640 data 240,   6, 202,  16, 248,  76, 189, 201, 189
IP  3650 data 253, 200,  10,  10,  10,  10, 141,  63, 192
GM  3660 data  44,  57, 192,  48,   6,  13,  62, 192, 141
MK  3670 data  63, 192,  32,  68, 199,  76, 189, 201,   5
BG  3680 data  28,  30,  31,  16,  28,  30,  31,   1,  21
BH  3690 data  22,  23,  24,  25,  26,  27,   1,   2,   5
NG  3700 data   6,   0,   4,   7,   3,   8,   9,  10,  11
AI  3710 data  12,  13,  14,  15, 201,  14, 208,   6,  32
PB  3720 data 245, 201,  76, 189, 201, 201,  17, 208,  11
IH  3730 data 162,  40,  32,  60, 201, 202, 208, 250,  76
FJ  3740 data 189, 201, 201,  18, 208,   8, 169,   0, 141
NB  3750 data  81, 192,  76, 189, 201, 201,  29, 208, 143
CP  3760 data  32,  60, 201,  76, 189, 201, 165, 253, 208
ML  3770 data   2, 198, 254, 198, 253, 165,   3, 208,   2
CK  3780 data 198,   4, 198,   3,  56, 165, 251, 233,   8
IM  3790 data 133, 251, 165, 252, 233,   0, 133, 252, 165
HJ  3800 data 251, 201,   0, 165, 252, 233, 224, 144,   1
GN  3810 data  96, 230, 253, 208,   2, 230, 254, 230,   3
FF  3820 data 208,   2, 230,   4,  24, 169,   8, 101, 251
IL  3830 data 133, 251, 144,   2, 230, 252, 165, 251, 201
DD  3840 data  64, 165, 252, 233, 255, 144,   3,  32,  60
MN  3850 data 201,  96,   9,  64, 174,  81, 192, 240,   2
JA  3860 data   9, 128,  32, 194, 201, 160,   7,  32,   3
CB  3870 data 202, 177,   5, 145, 251, 136,  16, 249,  32
LF  3880 data  18, 202, 200, 173,  62, 192,  44,  57, 192
PF  3890 data  16,   8, 173,  65, 192, 145, 253, 173,  64
CA  3900 data 192,  13,  63, 192, 145,   3,  32, 100, 201
MD  3910 data 104, 168, 104, 170,  96, 133,   5, 169,   0
IJ  3920 data 133,   6,   6,   5,  38,   6,   6,   5,  38
CN  3930 data   6,   6,   5,  38,   6,  24, 173,  72, 192
IO  3940 data 101,   5, 133,   5, 173,  73, 192, 101,   6
GI  3950 data 133,   6,  96,  32, 253, 174,  32, 138, 173
NI  3960 data  32, 247, 183, 166,  21, 208,   9, 165,  20
OF  3970 data 208,   3, 162, 208,  44, 162, 216, 142,  73
KH  3980 data 192, 162,   0, 142,  72, 192,  96, 173,  14
EF  3990 data 220,  41, 254, 141,  14, 220, 165,   1,  41
AI  4000 data 251, 133,   1,  96, 165,   1,   9,   4, 133
II  4010 data   1, 173,  14, 220,   9,   1, 141,  14, 220
HM  4020 data  96,  32, 121,   0, 240,  15,  32, 217, 193
GE  4030 data  32, 221, 194, 141,  77, 194, 142,  81, 194
HI  4040 data 169, 128,  36, 169, 141,  74, 192,  96, 255
EJ  4050 data   0, 223,   9, 239,   0, 255,   0, 255,  17
DM  4060 data 255,   0, 255,   0, 223, 202, 219,   9, 202
AG  4070 data   2, 219, 219
```

Listing 4: This program creates the file "hiprnt2.ml", also needed for the program in Listing 1.

```
OE  1000 rem* program to create file 'hiprnt2.ml' on disk *
LC  1010 for j=1 to 1136 : read x
BJ  1020 ch=ch+x : next
ON  1030 if ch<>145475 then print"checksum error" : end
FM  1040 print "data ok, now creating file": print
CD  1050 restore
OF  1060 open 8,8,8,"0:hiprnt2.ml,p,w"
EL  1070 print#8,chr$(0)chr$(152);
BH  1080 for j=1 to 1136 : read x
CM  1090 print#8,chr$(x); : next
ED  1100 close 8
AG  1110 print "prg file 'hiprnt2.ml' created. . .
KF  1120 print "this generator no longer needed.
II  1130 rem
IH  1140 data   1,   3,   7,  13,  15,  14,   4,  10,   8
LP  1150 data   5,  12,   6,   2,   9,  11,   0,   0,  68
JH  1160 data  17,  34, 102, 170, 221, 238, 119, 187, 255
```

```
LK  1170 data  10,  77,  27,  35,  27,  16,  51,  27,   0
BJ  1180 data 200,  75,  27,  13,  10,   1, 144,  75,  27
BJ  1190 data  13,  10,  10,  13,   0,  77,  27,  50,  27
JM  1200 data 217,   0, 152, 240,   3, 136,  16, 248,  96
MP  1210 data  32, 235, 194, 173,  83, 192, 240,   2, 169
MC  1220 data 255, 141,  81, 192, 169,  15, 168,  45,  62
AM  1230 data 192,  32,  54, 152, 173,  71, 192,  74,  74
CJ  1240 data  74,  74,  32,  54, 152, 152,  48,   2, 169
KM  1250 data   0, 172,  85, 192, 208,   2,  73, 255, 133
JA  1260 data   2, 169,  61,  32, 195, 255, 169,   0, 133
JC  1270 data 253, 133, 251, 169, 224, 133, 252, 169, 160
EA  1280 data 133, 254,  32,  20, 194, 160,   0, 177, 251
GB  1290 data 145, 253, 200, 208, 249, 230, 254, 230, 252
OB  1300 data 208, 243,  32,  46, 194, 173,  61, 192,  72
OD  1310 data  48,   3,  32,   7, 196, 169, 204, 133, 252
BK  1320 data 169, 190, 133, 254,  32, 109, 154, 169,  40
DC  1330 data 141,  75, 192,  32, 176, 154, 165,   1,  41
EC  1340 data 254, 133,   1, 174,  81, 192, 240,  11, 169
ID  1350 data  12,  32, 210, 255,  14,  75, 192,  76, 143
DI  1360 data 153, 169,  25, 141,  76, 192, 160,   5, 185
PE  1370 data  35, 152,  32, 210, 255, 136,  16, 247,  32
EJ  1380 data 235, 153,  44,  57, 192,  48,  46, 173,  79
FH  1390 data 192, 105, 150, 162,   0, 138,  42, 141,  77
EK  1400 data 192, 141,  78, 192,  10, 141,  79, 192,  13
HB  1410 data  77, 192, 141,  77, 192, 173,  80, 192, 105
HC  1420 data 150, 144,  11, 238,  78, 192, 238,  78, 192
KA  1430 data 238,  79, 192, 162,   3, 142,  80, 192, 160
FM  1440 data   7, 177, 253, 162,   4, 142,  81, 192,  32
AM  1450 data  92, 154,  72, 189,  77, 192, 106,  46,  83
OB  1460 data 192, 106,  46,  83, 192, 104, 206,  81, 192
JM  1470 data 208, 235, 173,  83, 192,  69,   2,  32, 210
LF  1480 data 255, 136,  16, 217, 206,  76, 192, 240,  23
AB  1490 data  56, 165, 253, 233,  64, 133, 253, 165, 254
MA  1500 data 233,   1, 133, 254, 173,  81, 192, 240, 138
JD  1510 data 208,  80,  76, 143, 154, 206,  75, 192, 240
LC  1520 data 248, 165, 197, 201,  63, 240, 242,  24, 173
EB  1530 data  81, 192, 170, 240,   7,  73, 128, 141,  81
CC  1540 data 192,  48,   6, 165, 253, 105,   8, 133, 253
EF  1550 data 165, 254, 105,  30, 133, 254, 202, 138,  56
MD  1560 data  48,   1,  24, 165, 251, 105, 232, 133, 251
KP  1570 data 165, 252, 105,   3, 133, 252, 232, 208,   3
GP  1580 data  76, 199, 152, 169,  25, 141,  76, 192, 160
NP  1590 data   5, 185,  41, 152,  32, 210, 255, 136,  16
IH  1600 data 247,  32, 235, 153, 160,   7, 177, 253,  44
HB  1610 data  81, 192,  16,   4,  10,  10,  10,  10,  44
NH  1620 data  57, 192,  48, 124, 162,   4,  10,  72, 173
BK  1630 data  79, 192, 144,   3, 173,  80, 192, 106,  46
OO  1640 data  83, 192, 106,  46,  83, 192, 104, 202, 208
NI  1650 data 234, 173,  83, 192,  69,   2, 170,  32, 210
CH  1660 data 255,  41,  85,  10, 141,  83, 192, 138,  41
KJ  1670 data 170,  74,  13,  83, 192,  32, 210, 255, 136
JO  1680 data  16, 188,  76,  54, 153,  56, 165, 251, 233
FH  1690 data  40, 133, 251, 176,   2, 198, 252, 160,   0
OA  1700 data 177, 251,  72,  41,  15, 170, 189, 103, 192
BP  1710 data 141,  79, 192, 104,  74,  74,  74,  74, 170
GP  1720 data 189, 103, 192,  44,  57, 192,  16,   3,  32
OP  1730 data  25, 154, 141,  80, 192,  96, 141,  78, 192
EP  1740 data 165, 252, 170,  41,   3,   9, 216, 133, 252
OO  1750 data 177, 251, 134, 252,  41,  15, 170, 189, 103
ED  1760 data 192,  96,  32,  92, 154,  72, 189,  77, 192
JP  1770 data  41, 240, 141,  83, 192, 104,  32,  92, 154
NH  1780 data 189,  77, 192,  41,  15,  13,  83, 192,  69
```

```
HC  1790 data   2, 170,  32, 210, 255,  41,  51,  10,  10
PD  1800 data 141,  83, 192, 138,  41, 204,  74,  74, 112
KB  1810 data 131, 162,   0, 142,  85, 192,  10,  46,  85
PD  1820 data 192,  10,  46,  85, 192, 174,  85, 192,  96
KD  1830 data 160,   5, 169,  61, 162,   4,  32, 186, 255
FF  1840 data 169,   0,  32, 189, 255,  32, 192, 255, 162
LB  1850 data  61,  32, 201, 255, 162,   7, 189,  27, 152
LC  1860 data  32, 210, 255, 202,  16, 247,  96, 162,   6
PL  1870 data 189,  47, 152,  32, 210, 255, 202,  16, 247
AM  1880 data 169,  61,  32, 195, 255,  32, 204, 255, 173
HJ  1890 data  60, 192,   9,   3, 133,   1, 104,  48,   3
KI  1900 data  32, 173, 195,  96, 162,  34, 169,   0, 157
MH  1910 data  84, 192, 202,  16, 250, 169,   3, 141,  83
BJ  1920 data 192, 160,   0, 177, 251,  72,  32,   7, 155
PL  1930 data 104, 174,  84, 192, 208,   8,  41, 240,  74
CK  1940 data  74,  74,  32,  10, 155, 136, 208, 233, 230
EN  1950 data 252, 206,  83, 192,  48,   6, 208, 222, 160
LO  1960 data 231, 208, 220, 174,  84, 192, 208,  42, 173
DO  1970 data  57, 192,  16,  37, 141,  84, 192, 173,  33
EM  1980 data 208,  41,  15,  10, 170, 189,  88, 192, 105
AC  1990 data   4, 157,  88, 192, 169, 216, 133, 252, 208
OO  2000 data 179,  41,  15,  10, 170, 254,  87, 192, 208
BA  2010 data   3, 254,  88, 192,  96, 169, 232, 133, 251
PO  2020 data 169, 207, 133, 252, 160,   0, 162,   0, 185
KK  2030 data  88, 192, 106, 141,  83, 192, 185,  87, 192
PA  2040 data 106, 110,  83, 192, 106,  44,  57, 192,  16
OP  2050 data   4, 110,  83, 192, 106, 105,   0, 157,  87
OO  2060 data 192, 200, 200, 232, 224,  16, 208, 220, 162
OB  2070 data   0, 142,  85, 192, 142,  83, 192, 169,   0
NJ  2080 data 141,  77, 192, 188,   0, 152, 185,  87, 192
FF  2090 data 201,  35, 176,  17, 109,  77, 192, 141,  77
CD  2100 data 192, 201,  35, 176,  12, 224,  15, 176,   8
LP  2110 data 232, 144, 229, 173,  77, 192, 208,   1, 232
OP  2120 data 142,  84, 192, 173,  85, 192, 238,  85, 192
BP  2130 data 174,  83, 192, 188,   0, 152, 153, 103, 192
IL  2140 data 232, 236,  84, 192, 144, 244, 224,  16, 144
OM  2150 data 187,  14,  85, 192,  14,  85, 192,  14,  85
BO  2160 data 192,  14,  85, 192, 202, 189, 103, 192,  44
CL  2170 data  85, 192,  48,   4,  10, 112,   1,  10, 168
KN  2180 data 185,  16, 152, 157, 103, 192, 202,  16, 234
FO  2190 data 162,  16, 202, 188,   0, 152, 185,  87, 192
HH  2200 data 208,   7, 169, 255, 153, 103, 192, 208, 240
IP  2210 data 185, 103, 192, 141,  83, 192, 169, 255, 153
IJ  2220 data 103, 192, 202, 188,   0, 152, 185, 103, 192
HG  2230 data 205,  83, 192, 240, 239, 173,  33, 208,  41
DD  2240 data  15, 170, 189, 103, 192, 141,  77, 192, 169
NK  2250 data   0, 141,  85, 192,  96, 162, 127, 160,  64
CN  2260 data 134, 252, 132, 251, 162, 204, 160,   0, 134
HH  2270 data 254, 132, 253, 162, 131, 160,  39,  32,  62
PN  2280 data 156, 162, 131, 160,  40, 134, 252, 132, 251
IM  2290 data 162, 216, 160,   0, 134, 254, 132, 253, 162
CP  2300 data 135, 160,  15,  32,  62, 156, 173,  16, 135
MG  2310 data 141,  33, 208,  32, 183, 193,  32, 230, 195
GH  2320 data 162,  96, 160,   0, 134, 252, 132, 251, 162
HO  2330 data 224, 160,   0, 134, 254, 132, 253, 162, 127
PE  2340 data 160,  63,  32,  62, 156,  96, 142,  78, 192
KL  2350 data 140,  77, 192, 160,   0, 177, 251, 145, 253
KP  2360 data 165, 252, 205,  78, 192, 208,   8, 165, 251
MJ  2370 data 205,  77, 192, 208,   1,  96, 230, 251, 208
EP  2380 data   2, 230, 252, 230, 253, 208, 227, 230, 254
AD  2390 data 208, 223,   0, 255,   0, 255,   0, 255,   0
NO  2400 data 255,   0
```

Computer Generated Holography on a C64

Patrick Hawley
Port Elgin, Ontario

. . .the key is knowing that diffraction effects are described by a Fourier Transform.

Introduction

In this programming project a Commodore 64 is used to produce a plot which can be photographically reduced to become a computer generated hologram.

When most people think of holograms, they think of the small pictures on their credit cards or the covers of National Geographic magazine (March/84 or Nov./85). These three-dimensional holograms are the record of an interference pattern made by a lensless photographic method that uses lasers. The interference pattern contains both the phase and amplitude information necessary to construct an image by the diffraction and interference of light. Computer generated holograms, such as the one that will be described in this article, are diffraction gratings (which is basically what a hologram is) that give two-dimensional diffraction patterns of an input image.

Background

In order to understand what the program does, a basic understanding of interference and diffraction of light waves is required. The term interference is used to describe the combining of two or more waves of the same frequency in the same region of space. Whether the interference is constructive or destructive depends on the phase relationship between the waves. If the waves are in phase their amplitudes will add; if the waves are out of phase, then their amplitudes will subtract. As will be explained, it is the constructive and destructive interference of light waves that makes the light and dark spots found in a diffraction pattern.

Diffraction is the term used to name a phenomenon that light possesses, which is that it spreads out when passing an edge. An explanation for this can be given in terms of interference effects within a single beam when the light beam is thought of as the sum of many individual sources (Huygen's principle). Diffraction effects are most noticeable when dealing with narrow beams of light when the effect of the light spreading around an edge means that geometric optics can no longer be applied.

Insight can be gained into interference, diffraction and diffraction gratings by considering the case of a beam of light passing through two slits as shown in Figure 1. The lines showing the light represent a specific phase of the wave (the wave crests if you like). As the waves hit the slits part of them will go through. On the "downstream" side of the slits, the light spreads out in all directions due to diffraction. This causes the two beams coming from the slits to overlap. If a screen is placed at a distance that is far compared to the distance between the slits, a diffraction pattern can be seen. Wherever the overlapping beams interfere constructively, the screen will be bright. Wherever the overlapping beams interfere destructively, the screen will be dark. For example, if the two beams must travel the same distance to reach a given point on the screen, they will arrive there in phase and make a bright spot. If one beam must travel half a wavelength more than the other to reach a given point, it will arrive out of phase, and that point will be dark. These regions of alternating bright and dark intensity are called interference fringes and the whole image on the screen is the interference or diffraction pattern. The diffraction effects seen on a screen that is far compared to the distance between the slits (as in this case) are called Fraunhofer diffraction.

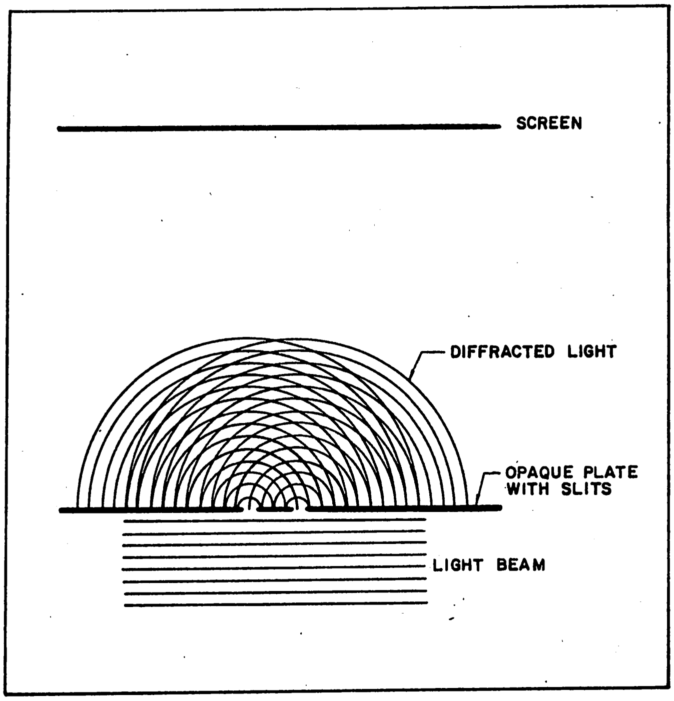


Figure 1: Light diffracted by a double slit. A distant screen would show a diffraction pattern of light and dark spots corresponding to areas of constructive and destructive interference.

Now the concept of the computer program can be explained. We want a program that will arrange slits in such a way that when light is diffracted by the slits the resulting diffraction pattern forms a predetermined input image. Our arrangement of slits can then be called a hologram.

To go further we must know something about the mathematical description of light. Light is an electromagnetic wave which can be described using complex numbers. The key to constructing the program is knowing that the effect of Fraunhofer diffraction is described by a Fourier transformation.

Fourier transforms are themselves the subject of books. Briefly, what Fourier transforms do is distinguish the frequency components in a varying signal. (Conversely, inverse Fourier transforms give the domain of a signal whose frequency components are known). The proof that Fourier transformations describe Fraunhofer diffraction and that they can be used as described here to generate holograms goes beyond the scope of this article. For those interested, the proof may be found in a paper by Lohmann and Paris in Applied Optics, Vol. 6, No. 10, Pg. 1739. The essential thing to know is that the Fourier transform will provide the information required to tell us how to position our slits to make the diffracted light hit the screen at the right phase and how big to make the slits so that the light reaches the screen with the right amplitude.

Method

To make the slits, a printer or plotter is used to draw small black rectangles on a piece of paper. When this page is photographed, the black rectangles are clear on the negative and we have our slits. The details of the procedure are as follows:

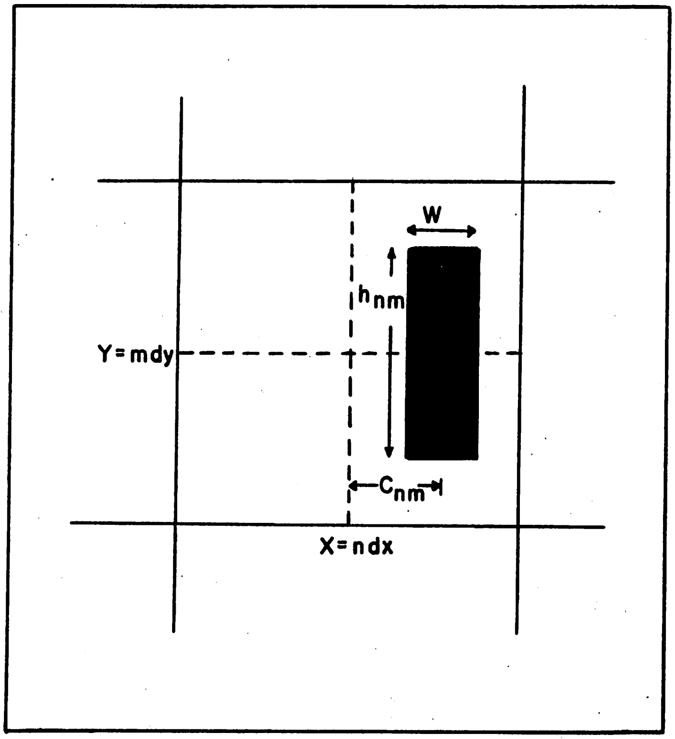


Figure 2: Typical aperture in a cell. The height, h_{nm}, encodes amplitude information and the position, C_{nm}, encodes phase information.

The desired input image is drawn on a grid. The input data is an array of binary elements made up by assigning a zero to any part of the grid which doesn't make part of the image and a one to any part of the grid which does make part of the image. The input data is transformed using a two-dimensional complex Fourier transform. The coefficients of the transform are then used to calculate the information required to make a plot of small black rectangles on a page.

To make the plot, the paper is divided into equally spaced cells. Rectangular apertures (i.e. the slits) are drawn inside each cell. Each aperture has three parameters: its height, 'hjnmt', its width, 'w', and its centre with respect to the centre of the cell, 'cjnmt'. The subscripts, nm, indicate to which cell on the page the aperture belongs. Figure 2 shows one of the cells and aperture. The height of the aperture is made proportional to the calculated amplitude and the position of the aperture is made proportional to the calculated phase.

When the plot is reduced down on film to become a hologram, and monochromatic, coherent light is shone through, we get a Fraunhofer diffraction pattern which is a reconstruction of our original input image. The hologram is now doing the inverse of the original Fourier transform which we applied to the input data.

The Program

The first part of the program asks for some input. The first option is whether or not you want to make a hologram plot. At this time I should point out that aside from producing the hologram plot the program also has the feature that it continues on and works backwards, starting with the final data used to create the plot, to reconstruct the original input image. The image is then drawn with the printer using a gray-scale made up from ten characters available from the Commodore keyboard. I found this feature very useful for checking my input data, debugging the program, and for getting a qualitative feeling for the image quality I might expect without going through the trouble of photographing a plot. If you select the option of not doing the plot, you still get the gray-scale reconstructed image.

Next, you are asked if you want to keep the phase of the input elements constant or if you want them randomized (but still binary). Explanation: The two parts of the complex input data give the real and imaginary components of the input wave. The phase is given by the arctan of the imaginary part over the real part. The magnitude is given by the square root of the sum of the squares of the real and imaginary parts. Thus the input phase can be made random while keeping the magnitude constant so the input remains binary. If you keep the input phases constant, the dynamic range of the transform will be quite large, with a few of the elements disproportionately larger than most. The result of this will be that after normalization, most of the amplitudes will be small and your plot will be mostly of tiny rectangles which in turn will give a dim image. This can be overcome somewhat by truncating the larger coefficients and if you choose a constant phase input the program will scan the transform and help you decide how much truncation you want to do. It's fun to try to optimize the final image with different amounts of truncation using the gray-scale output to make qualitative judgements.

If you randomize the phase of the input data, the dynamic range of the transform will be reduced and the distribution of the amplitudes more uniform. This gives the best final image.

After these decisions have been made, the matrix which makes up the input figure is read. As can be seen in the listing, the ones in the 32 X 32 matrix being read form the image of a "happy face". This data is used to input values to the arrays AR(D,E) and AI(D,E) which represent the real and imaginary components of the "happy face" wavefront. The choice of an imput image must be made keeping in mind that the final image quality is dependent on the number of sampling points used to mathematically represent the input wave (i.e. the number of points in the grid on which the original figure was drawn). The more points available, the more complex the figure can be, however, the figure should be simple enough so that it can be easily represented within the given matrix size.

There are other restrictions as well. Since the results of sampling from the grid have to be read into our computer, the grid size is limited to arrays which can fit into the available computer memory. Also, the Fourier transform is performed using a so called Fast Fourier Transform algorithm which requires that the matrix dimensions must be m X n where m and n are an integral power of two. The Fast Fourier Transform used here I adapted from a FORTRAN listing by D.E. Jones in a paper that Jones wrote while a student at University of Toronto.

These are the considerations which led to a matrix size of 32 X 32 and a "happy face" figure. Other appropriate input figures for this size matrix could be letters of the alphabet, Chinese characters, the symbol for peace, or whatever.

After the Fourier transform has been completed, the coefficients of the transform are used to calculate an amplitude and phase for each component. The amplitudes are then scanned, and the largest is used to normalize the amplitudes to between zero and one (unless a constant phase was used in the input data, in which case there is an option to normalize with a smaller number and truncate the larger amplitudes to one). We now have the required information to make the plot.

I will describe the plot routine in some detail since this is the part of the program that you will probably be modifying should you want to follow through on this article and try this on your own computer. The way you choose to make your plot will affect the final image quality so it's best to have some guidelines before starting. First I will describe how the plot routine listed here works, then I will discuss what to consider should you be using a plotter.

I produced the plot on a Commodore 1526 printer. Each cell is two printer characters (16 pixels) wide and two printer characters (16 pixels) tall. The rectangles to be placed inside the cells are four pixels wide. Avoiding the complication of having rectangles overlap at cell edges, this allows for a maximum of 13 horizontal positions (quantized phase values) and nine rectangle heights (quantized amplitude values) if you count zero as one of the heights.

To plot specific pixels with a 1526 printer, you must define custom characters. Thus to make the plot, I first set up an array, CH$(I,J), which contained all the different characters I would need to make the apertures of varying heights and positions within a cell. The large number of characters required, and the complexity of the plot routine, is mainly due to the fact that each cell is made up of four (2 X 2) printer characters.

In the two-dimensional array, CH(I,J), the first indicia keeps track of characters of different horizontal positions while the second keeps track of characters of different height. The correct height indicia is found by multiplying the normalized amplitude by eight (since on the 1526 printer, characters are eight pixels tall) and determining to which of the nine quantized values the amplitude belongs. A similar process is used to find the horizontal indicia except that now the phase is used to find the offset from the middle of the cell.

It takes two rows of characters to make one row of rectangular apertures. First the top half of a row of apertures is printed, then the bottom. The plot routine keeps track of which half is currently being printed and if it's the bottom half, the character height indicia is shifted to print characters of the proper size which meet the bottom of the top half of the rectangle and grow downwards. You may also note that the 1526 printer can handle only one custom character at a time and has a one line buffer. This explains the need for all the tabbing and carriage returns.

By now some of you must be thinking that there must be a better way. I'm sure there is. For example, those who have systems that can provide the required resolution may be able to do it neatly on a hi-res screen which is then simply dumped on a printer or those with plotters may have another way. At any rate, here are some guidelines which can be used when setting up a plot routine. The guidelines were drawn by reading the previously mentioned paper by Lohman and Paris and from a paper by Gabel and Liu in Applied Optics, Vol. 9, No. 5, Pg. 1180.

The image brightness is maximized when the relative dot width is one half the cell width and the full width of the cell is used for dot positioning. If you use the full cell width, you must write a plot routine which will allow the dots to overlap into neighbouring cells. There will be some error in the odd cases where two dots overlap because these are areas which should be doubly transmissive. In practise though, there won't be enough of these cases to worry about. Lohman and Paris also showed however, that image brightness range increases with increasing dot width. They suggest that a good balance between image brightness and uniformity might be achieved with a dot width of one third the cell width.

Gabel and Liu discuss minimizing image reconstruction error by optimizing the number of sample points (cells on the plot) and quantized steps available within a cell given the total number of steps available on your plotter. It turns out that the plot I was able to do on my printer in which I had up to 16 steps within each of 32 X 32 cells was close to being an optimum arrangement.

How far the images end up being spaced from the optic axis can be controlled by how much of the cell width is used for aperture positioning. The smaller the amount used, the farther away from the optic axis the images will be.

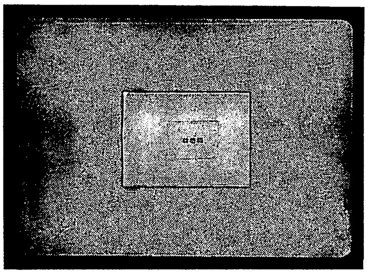
Some other things to keep in mind are that amplitude errors (aperture height) aren't as important as phase errors (aperture position) because they do not deviate the light rays as do phase errors. Also, if your printer or plotter is limited, more cells or finer quantization can be had by doing the plot in several parts which can later be pasted together before the reduction.

Running, Reduction, Reconstruction

Once the program starts running, be prepared to wait. It takes over an hour for the program to run completely. I put in some print statements to the monitor so you can follow along and see where the program is at any given time.

The output plot must be reduced so that the distance between apertures is in the order of tens of microns. This means that for a 32 X 32 plot which is 15 cm^2 a reduction factor of about 100 is good.

The reduction is done photographically. None of the authors of the papers I've read seem to think that there's any challenge to this process and give no details on how it's done. I, with the help of a friend who's into photography, tried a few times with various low grain, high contrast films but we couldn't seem to get the exposure right. Either the apertures were filled in or the background was too light. If anybody out there has some suggestions, I'd love to hear them. (My mailing address is PO Box 5106, Port Elgin, Ontario, N0H 2C0.) I finally sent my plots to a microfiching company who reduced them for $20.00 (see below). Even the best they could do was a reduction factor of 79.



The three tiny white squares (actual size) at the center of the film are three separate hologram plots.

In order to reconstruct the image with the hologram we need a light source which can display interference effects. This means that it must be monochromatic and coherent. These properties can be found in the light coming from a laser. What!? You don't own a laser? Not surprising, but don't be disheartened. Most high schools these days have relatively inexpensive helium neon lasers and it's been my experience that physics teachers are delighted to let you borrow their laser provided you let them see what you've done. If you're a student, make a computer-generated hologram, take it into class and knock your teacher's socks off.

A 15 cm^2 plot reduced by a factor of 100 ends up as a 1.5 mm^2 hologram. This is convenient because it's about the same as the beam diameter of an inexpensive laser. When the laser beam is shone through the hologram, images of the input will appear on a screen placed on the other side. The images closest to the optic axis are first order diffraction, the next set of images are second order, and so on.

Results

Figure 3 shows typical result of running the program. The input was binary data that formed a "happy face". Figure 3(a) is the plot produced by a 1526 printer. Figure 3(b) is the gray-scale reconstruction of the image, also done on the 1526. Figure 3(c) is a photograph of the diffraction pattern seen when a laser beam is shone through the hologram. Notice the dots which appear in the gray-scale picture and the reconstructed image. These dots are from Fourier transforming "square" data such as the quantized data we use to make our plot.

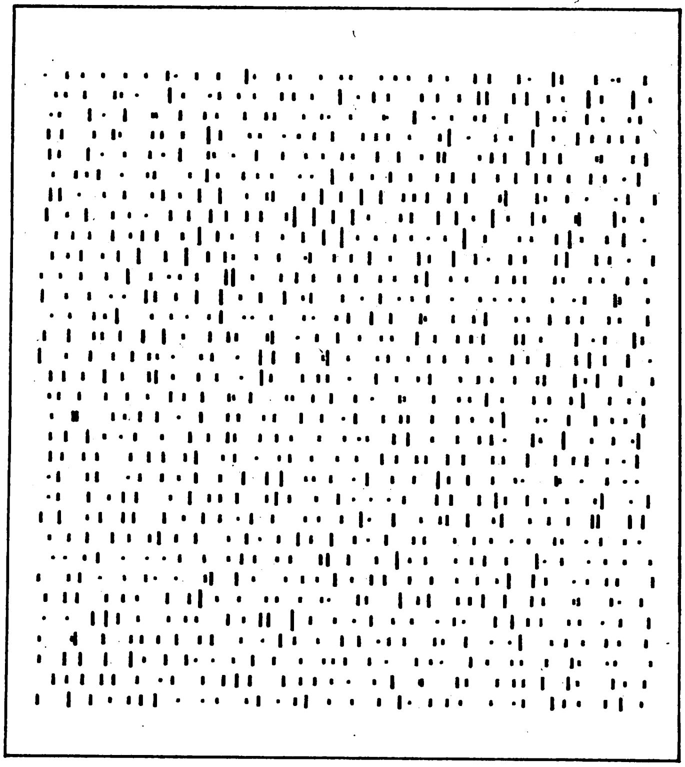


Figure 3(a): Hologram plot from 1526 printer (54% of actual size). Reduced photographically, the above will become a diffraction grating which will form images of "happy faces".

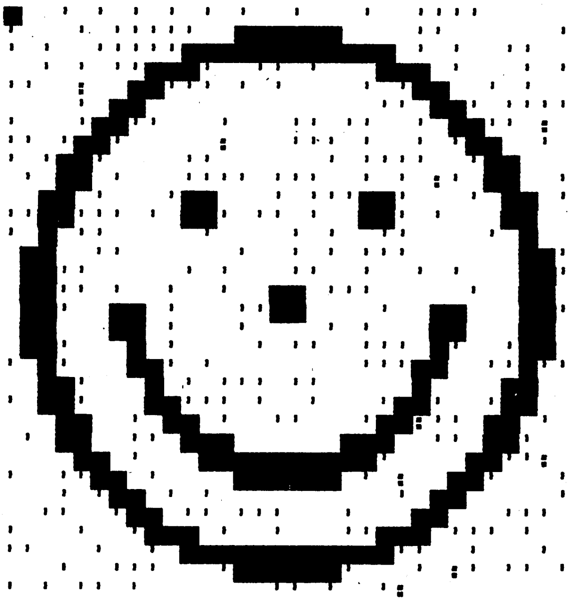


Figure 3(b): Gray scale reconstruction of image corresponding to Figure 3(a).

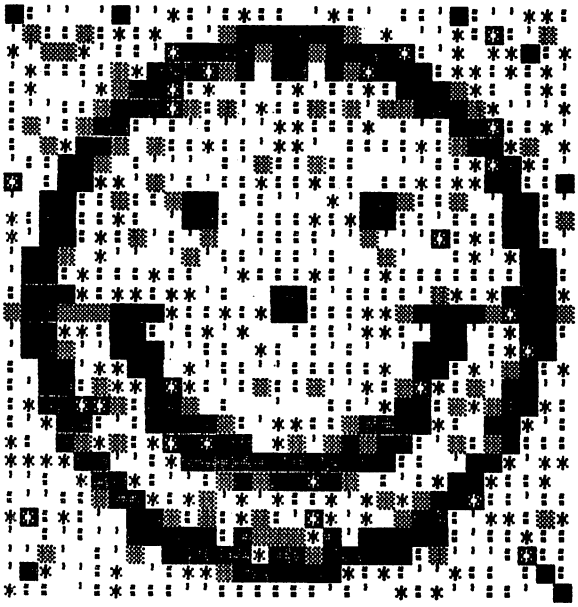


Figure 4(a): Gray scale reconstruction of an image that used a constant input phase and no truncation of the transformed data.

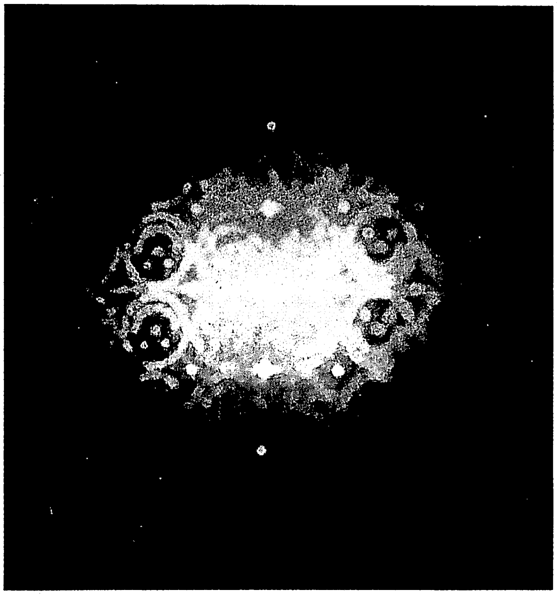


Figure 3(c): Photograph of the actual diffraction pattern produced by shining a laser beam through the photo-reduced plot on film.

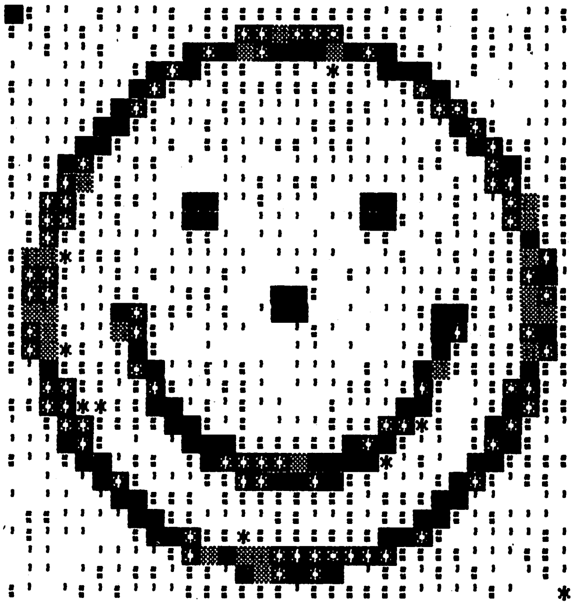


Figure 4(b): Gray scale reconstruction of an image that had a constant input phase. The transformed data was truncated to limit the effect of the dominant coefficients which result from transforming square data.

Figure 4 shows gray-scale pictures which show the effect of using a constant input phase and truncating the transformed data. In Figure 4(a) no truncation was performed and as can be seen, the predicted reconstructed image will be of relatively poor quality. In Figure 4(b) the few dominant transform amplitude coefficients were truncated and the gray-scale picture predicts a much better image.

There are other methods of making computer generated holograms. Some record phase information on film by varying the emulsion thickness. The binary holograms described here are the easiest to make, with the required computer hardware probably already being available to the majority of persons who have read this article.

Computer Generated Holograms

```
CF  10 rem this program makes binary holograms
LH  15 rem by: patrick hawley, port elgin, on
IB  20 dim x(32),y(32),s(13),u(13),ar(32,32),ai(32,32),ch$(11,15),e(16)
OC  30 ti$="000000":open4,4
AD  40 ir=1: ii=1
LM  50 read n,it
CE  60 print""
OJ  70 input"do you want a hologram plot (y/n)";hp$
ND  80 input"random or constant phase (r/c)";ip$
ME  90 pq=13
NF  100 for d=1 to n
KG  110 for e=1 to n
CA  120 read ar(d,e)
AI  130 ai(d,e)=ar(d,e)*ii
AM  140 ar(d,e)=ar(d,e)*ir
HL  150 ifip$="c"then200
HI  160 ar(d,e)=(-1+2*rnd(1))*sqr(ir↑2+ii↑2)*ar(d,e)
MG  170 pm=int(-1+3*rnd(1))
AF  180 if pm=0 then 170
KF  190 ai(d,e)=sqr(ir↑2+ii↑2-ar(d,e)↑2)*ai(d,e)*pm
PI  200 next e,d
BN  220 data 32,1
EB  230 data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
KC  240 data 0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0
AE  250 data 0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0
OD  260 data 0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0
EE  270 data 0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0
OE  280 data 0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0
IF  290 data 0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0
CG  300 data 0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0
MG  310 data 0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0
GH  320 data 0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0
II  330 data 0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0
CJ  340 data 0,0,1,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0
AJ  350 data 0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0
OJ  360 data 0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0
IK  370 data 0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0
GL  380 data 0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0
IM  390 data 0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0
OM  400 data 0,1,1,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,0
EN  410 data 0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0
ON  420 data 0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0
IO  430 data 0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,0,0
GP  440 data 0,0,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0
AA  450 data 0,0,0,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,1,0,0,0
OA  460 data 0,0,0,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,0,0,0
AC  470 data 0,0,0,0,1,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,1,1,0,0,0,0
CC  480 data 0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0
AC  490 data 0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0
KC  500 data 0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0
ID  510 data 0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0
OE  520 data 0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0
ME  530 data 0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0
KE  540 data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
KJ  550 :
AD  560 gosub 660:rem fast fourier transform
OA  570 gosub 2130:rem amplitude and angle
GP  580 gosub 2370:rem normalize/truncate
JG  590 gosub 2900:rem hologram plot
HO  600 gosub 4340:rem find components for inverse fft from plot data
FM  610 it=-1
BK  620 gosub 660:rem inverse fft
IB  630 gosub 4430:rem gray-scale plot
GA  640 print"finished. time is ";ti$
KI  650 end
GC  660 rem 2d fft
KC  670 for rw=1 to n
DI  680 print"[DN] working on fft;row:";rw
GP  690 for re=1 to n
PJ  700 x(re)=ar(rw,re):y(re)=ai(rw,re)
EN  710 next re
GK  720 gosub 890
OB  730 for re=1 to n
HG  740 ar(rw,re)=x(re):ai(rw,re)=y(re)
BN  750 next re,rw
FD  770 for cl=1 to n
BC  780 print"[DN] working on fft;column:";cl
NC  790 for ce=1 to n
PI  800 x(ce)=ar(ce,cl):y(ce)=ai(ce,cl)
LA  810 next ce
KA  820 gosub 890
FF  830 for ce=1 to n
PL  840 ar(ce,cl)=x(ce):ai(ce,cl)=y(ce)
JM  850 next ce,cl
CI  870 return
EO  880 :
FO  890 rem  complex fast fourier transform
IP  900 :
NE  910 if it>0 then 940
AK  920 for i=1 to n
DB  930 y(i)=-y(i): next i
JA  940 lg=log(n)/log(2)
EB  950 if lg<=1 then 1390
CP  960 for k=2 to lg step 2
MP  970 m=2↑(lg-k)
FE  980 m4=4*m
GO  990 for j=1 to m
DP  1000 ar=(j-1)/m4*2*π
FB  1010 c1=cos(ar)
AE  1020 s1=sin(ar)
IM  1030 c2=c1*c1-s1*s1
AA  1040 s2=c1*s1+c1*s1
FO  1050 c3=c2*c1-s2*s1
NC  1060 s3=c2*s1+s2*c1
LI  1070 for i=m4 to n step m4
AA  1080 j0=i+j-m4
NA  1090 j1=j0+m
MB  1100 j2=j1+m
LC  1110 j3=j2+m
AP  1120 r0=x(j0)+x(j2)
PP  1130 r1=x(j0)-x(j2)
GP  1140 i0=y(j0)+y(j2)
FA  1150 i1=y(j0)-y(j2)
EC  1160 r2=x(j1)+x(j3)
DD  1170 r3=x(j1)-x(j3)
KC  1180 i2=y(j1)+y(j3)
JD  1190 i3=y(j1)-y(j3)
CC  1200 x(j0)=r0+r2
PO  1210 y(j0)=i0+i2
DN  1220 if ar=0 then 1300
LG  1230 x(j2)=(r1+i3)*c1+(i1-r3)*s1
OJ  1240 y(j2)=(i1-r3)*c1-(r1+i3)*s1
HI  1250 x(j1)=(r0-r2)*c2+(i0-i2)*s2
HK  1260 y(j1)=(i0-i2)*c2-(r0-r2)*s2
PJ  1270 x(j3)=(r1-i3)*c3+(i1+r3)*s3
AN  1280 y(j3)=(i1+r3)*c3-(r1-i3)*s3
```

```
BB   1290 goto 1360
CH   1300 x(j2)=r1+i3
HH   1310 y(j2)=i1-r3
NJ   1320 x(j1)=r0-r2
KG   1330 y(j1)=i0-i2
NJ   1340 x(j3)=r1-i3
OJ   1350 y(j3)=i1+r3
CG   1360 next i,j,k
BB   1390 if lg=int(lg/2)*2 then 1500
GF   1400 for i=1 to n step 2
CC   1410 r0=x(i)+x(i+1)
AD   1420 r1=x(i)-x(i+1)
HC   1430 i0=y(i)+y(i+1)
FD   1440 i1=y(i)-y(i+1)
JD   1450 x(i)=r0
KC   1460 y(i)=i0
NA   1470 x(i+1)=r1
AB   1480 y(i+1)=i1
BL   1490 next i
CK   1500 s(13)=n/2
PJ   1510 u(13)=n
AH   1520 for k=2 to 12
DN   1530 j=14-k
KG   1540 s(j)=1
DF   1550 u(j)=s(j+1)
OO   1560 if s(j+1)>1 then s(j)=int(s(j+1)/2)
HA   1570 next k
HD   1580 al=s(2)
AK   1590 jj=0
AB   1600 a=1: b=a: c=b: d=c: e=d: f=e
JC   1660 for g=f to u(7) step s(7)
KD   1670 for h=g to u(8) step s(8)
LE   1680 for i=h to u(9) step s(9)
BE   1690 for j=i to u(10) step s(10)
FF   1700 for k=j to u(11) step s(11)
JG   1710 for l=k to u(12) step s(12)
NH   1720 for m=l to u(13) step s(13)
MJ   1730 jj=jj+1
GH   1740 if jj<=m then 1810
CD   1750 t=x(jj)
PH   1760 x(jj)=x(m)
PM   1770 x(m)=t
EF   1780 t=y(jj)
DK   1790 y(jj)=y(m)
PO   1800 y(m)=t
GI   1810 :
OI   1820 next m,l,k,j,i,h,g
FO   1890 f=f+s(6)
EE   1900 if f<=u(6) then 1660
PO   1910 e=e+s(5)
AF   1920 if e<=u(5) then 1650
JP   1930 d=d+s(4)
MF   1940 if d<=u(4) then 1640
DA   1950 c=c+s(3)
IO   1960 if c<=u(s) then 1630
NA   1970 b=b+s(2)
EH   1980 if b<=u(2) then 1620
OG   1990 a=a+1
JB   2000 if a<=al then 1610
EJ   2010 for sl=1 to (n-2)/2
HG   2020 bc=n+1-sl:fc=sl+1
CH   2030 tx=x(fc):ty=y(fc)
AA   2040 x(fc)=x(bc):y(fc)=y(bc)
FL   2050 x(bc)=tx:y(bc)=ty
JD   2060 next sl
```

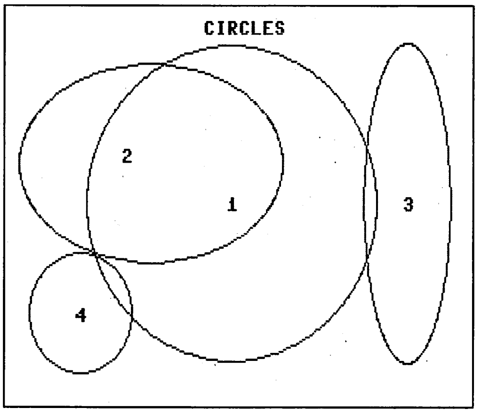
```
FJ   2070 if it>0 then return
IC   2080 for i=1 to n
FI   2090 x(i)=x(i)/n
LK   2100 y(i)=-y(i)/n
NB   2110 nexti
EG   2120 return
EO   2130 rem convert into amplitude & angle
NG   2140 print"...finding amplitude & angle"
PF   2150 for d=1 to n
MG   2160 for e=1 to n
ML   2170 ap=sqr(ar(d,e)*ar(d,e)+ai(d,e)*ai(d,e))
FJ   2180 if ap=0 then 2330
GP   2190 if ar(d,e)<>0 then 2230
LB   2200 if ai(d,e)<0 then pa=-π/2
DJ   2210 if ai(d,e)>0 then pa=π/2
CL   2220 goto 2320
KA   2230 if ai(d,e)<>0 then 2270
GD   2240 if ar(d,e)<0 the npa=π
CJ   2250 if ar(d,e)>0 thenpa=0
KN   2260 goto 2320
KJ   2270 pa=atn(ai(d,e)/ar(d,e))
PP   2280 if ar(d,e)>0 and ai(d,e)>0 then pa=pa
HC   2290 if ar(d,e)<0 and ai(d,e)>0 then pa=π+pa
KM   2300 if ar(d,e)<0 and ai(d,e)<0 then pa=pa-π
FC   2310 if ar(d,e)>0 and ai(d,e)<0 then pa=pa
JG   2320 ar(d,e)=ap: ai(d,e)=pa
BO   2330 next e,d
KE   2350 return
MK   2360 :
KB   2370 rem amplitude normalization routine
CB   2380 rem find largest amplitude
DK   2390 la=ar(1,1)
JF   2400 for d=1 to n
GG   2410 for e=1 to n
IL   2420 ifar(d,e)>lathenla=ar(d,e)
FE   2430 next e,d
BH   2450 if ip$="r" then 2820
JN   2460 print"the largest amplitude is";la
IF   2470 rem check amplitude distribution
LC   2480 p8=0: p6=0: p4=0: p2=0: p1=0
MC   2490 n8=.8*la:n6=.6*la:n4=.4*la:n2=.2*la:n1=.1*la
EO   2500 print" "
HM   2510 for d=1 to n
EN   2520 for e=1 to n
BF   2530 if ar(d,e)<=n8 then p8=p8+1
LE   2540 if ar(d,e)<=n6 then p6=p6+1
FE   2550 if ar(d,e)<=n4 then p4=p4+1
PD   2560 if ar(d,e)<=n2 then p2=p2+1
BE   2570 if ar(d,e)<=n1 then p1=p1+1
LN   2580 next e,d
DB   2600 p8=int(p8/1024*100):p6=int(p6/1024*100)
          :p4=int(p4/1024*100)
JB   2610 p2=int(p2/1024*100):p1=int(p1/1024*100)
PA   2620 print"80% of the max. amp is";n8
BJ   2630 print p8;"% of amp's are equal or smaller"
AH   2640 print" "
NB   2650 print"60% of the max. amp is";n6
HK   2660 printp6;"% of amp's are equal or smaller"
OI   2670 print" "
LC   2680 print"40% of the max. amp is";n4
NL   2690 printp4;"% of amp's are equal or smaller"
MK   2700 print" "
JD   2710 print"20% of the max. amp is";n2
DN   2720 printp2;"% of amp's are equal or smaller"
KM   2730 print" "
```

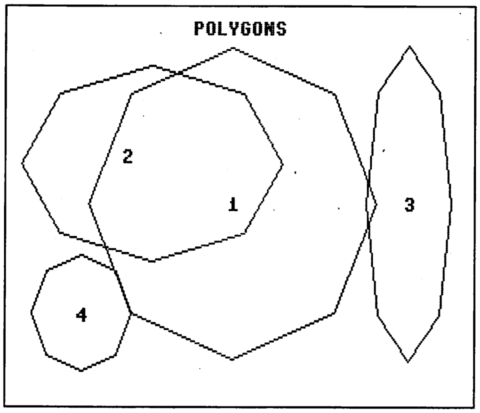
```
PE  2740 print"10% of the max. amp is";n1
NO  2750 printp1;"% of amp's are equal or smaller"
IO  2760 print" "
JL  2770 print"time is    ";ti$
MP  2780 print" "
DK  2790 input"what no. do you want to normalize to";nl
AB  2800 print" "
CH  2810 input"which means what % of data truncated";pt
CA  2820 ifip$="r"thennl=la
HA  2830 for d=1 to n
EB  2840 for e=1 to n
EI  2850 ar(d,e)=ar(d,e)/nl: if ar(d,e)>1 then ar(d,e)=1
DP  2860 next e,d
MF  2880 return
OL  2890 :
AB  2900 rem plot routine
ED  2910 rem read in plot characters
DD  2920 if hp$="n" then 3720
EO  2930 print". . .defining plot characters"
PO  2940 for i=1 to 8:e(i)=2↑i-1:e(i+8)=e(i)*2↑(8-i):next
DM  2950 for i=1 to 11: for j=1 to 15: for k=1 to 8
KI  2960 ch=0: if k<(13-i) and k>(8-i) then ch=e(j)
DE  2970 ch$(i,j)=ch$(i,j)+chr$(ch)
GL  2980 next k,j,i
CP  3710 :
HD  3720 open 2,4,5
PJ  3730 open 6,4,6:print#6,chr$(21)
MK  3740 rem main plotting loop
MA  3750 tb=0
PH  3760 nf=2: if hp$="n" then nf=1
DL  3770 for d=1 to n
LN  3780 for f=1 to nf
KM  3790 for e=1 to n
IL  3800 rem determine which character
NH  3810 rem find character height
ME  3820 ht=8*ar(d,e)
LD  3830 dh=ht-int(ht)
KL  3840 if dh<.125/2 then ht=int(ht):goto3860
LJ  3850 ht=int(ht)+1
OG  3860 if f=nf then ar(d,e)=ht/8*nl
OM  3870 rem find phase
ON  3880 cp=pq*ai(d,e)/(2*π)
BF  3890 qp=int(pq/2)
DD  3900 ifabs(cp)-int(abs(cp))>.5then3950
LI  3910 if cp<0 then cp=int(cp):if f=nf then
             ai(d,e)=(cp+1)*π/qp
DN  3920 if cp>=0 then cp=int(cp)+1:if f=nf then
             ai(d,e)=(cp-1)*π/qp
LB  3930 if hp$="n"then 4290
LI  3940 goto 3980
EJ  3950 if cp<0 then cp=int(cp)-1:if f=nf then
             ai(d,e)=(cp+1)*π/qp
NJ  3960 if cp>0then cp=int(cp)+2:if f=nf then
             ai(d,e)=(cp-1)*π/qp
DE  3970 if hp$="n" then 4290
AE  3980 if f<>1 then 4230
NP  3990 if ht=0 then print#4,"[2 spcs]";chr$(141);:goto4260
NA  4000 if abs(cp)>2then 4130
FP  4010 if cp>-2 then 4050
NJ  4020 print#2,ch$(3,ht):print#4,chr$(254);chr$(141);
IM  4030 tb=tb+1:print#2,ch$(11,ht):print#4,tab(tb);
             chr$(254);chr$(141);
EN  4040 goto 4260
LP  4050 if cp>1then 4090
EM  4060 print#2,ch$(2,ht):print#4,chr$(254);chr$(141);
```

```
OO  4070 tb=tb+1:print#2,ch$(10,ht):print#4,tab(tb);
             chr$(254);chr$(141);
MP  4080 goto 4260
BO  4090 print#2,ch$(1,ht):print#4,chr$(254);chr$(141);
JD  4100 tb=tb+1:print#2,ch$(9,ht):print#4,tab(tb);
             chr$(254);chr$(141);
DE  4110 tb=tb+1:print#2,ch$(9,ht):print#4,tab(tb);
             chr$(254);chr$(141);
EC  4120 goto 4260
KE  4130 if cp<0 then 4180
PK  4140 cp=cp+11-cp*2
OG  4150 print#4,"[1 spc]";chr$(141);
KJ  4160 tb=tb+1:print#2,ch$(cp,ht):print#4,tab(tb);
             chr$(254);chr$(141);
GF  4170 goto4260
BH  4180 if cp>0 then 4230
LL  4190 cp=abs(cp)+1
HF  4200 print#2,ch$(cp,ht):print#4,chr$(254);chr$(141);
MC  4210 tb=tb+1:print#4,tab(tb);"[1 spc]";chr$(141);
II  4220 goto 4260
CN  4230 if ht=0 then 3990
AA  4240 if ht<8 then ht=ht+8
CM  4250 goto 3990
NI  4260 if e=n then print#4,chr$(141);chr$(13);
FJ  4270 tb=2*e:if e=n then tb=0
FB  4280 if e<>n then print#4,tab(tb);
PK  4290 next e,f,d
MP  4320 return
OF  4330 :
HN  4340 rem find real and imaginary components from
             truncated & quantized data
EA  4350 for d=1 to n: for e=1 to n
KO  4370 ap=ar(d,e)
PM  4380 ar(d,e)=ap*cos(ai(d,e))
PL  4390 ai(d,e)=ap*sin(ai(d,e))
HP  4400 next e,d
AG  4420 return
CM  4430 :
FP  4440 rem gray scale picture routine
EE  4450 print#4,"[1 spc]"
FG  4460 for d=1 to n
BH  4470 print#4,tab(23);
MH  4480 for e=1 to n
GN  4490 in=sqr(ar(d,e)*ar(d,e)+ai(d,e)*ai(d,e))
BE  4500 af=1/sqr(ir*ir+ii*ii):in=af*in
ID  4510 if in>1 then in=1
OI  4520 if in<.1 thenprint#4,"[1 spc]";:goto4590
AN  4530 if in<.2 thenprint#4,"'";:goto4590
DP  4540 if in<.4 thenprint#4,":";:goto4590
DP  4550 if in<.6 thenprint#4,"*";:goto4590
KG  4560 if in<.8 thenprint#4,chr$(166);:goto4590
KF  4570 if in<.9 thenprint#4,"[RVS]*[OFF]";:goto4590
HO  4580 if in<=1 thenprint#4,"[RVS][1 spc][OFF]";
LP  4590 if e=n then print#4,chr$(13);
PL  4600 next e,d
OO  4620 print#4,"[1 spc]"
JK  4630 print#4,"input real amplitude:";ir
LM  4640 print#4,"input imaginary amplitude:";ii
JI  4650 print#4,"phase angle:";ip$
FB  4660 print#4,"no. of phase quantization levels:";pq
NB  4670 if ip$="r" then 4700
DL  4680 print#4,"transform normalized with. . .";nl
LJ  4690 print#4,"percent of transformed amp's
             truncated. . .";pt
IH  4700 return
```

CIRCLES for the C64

Anthony Bryant
Winnipeg, Manitoba





Every graphics programmer needs to draw circles. Fast assembly code circles are at best a challenge and at worse a headache. It is an opportunity to use a variety of integer math algorithms, testing one's latest GW (gee-whiz) super fast routines.

The program CIRCLES.BAS demonstrates two of the best known circle-drawing algorithms. It uses HIRES by Gary Kiziak (Vol.5 Iss.6 of the Transactor, reprinted in Vol.8 Iss.2) for the underlying graphics primitives. It's in Basic, so it's slow, but also easier to study, and then translate to assembly code.

Polygons

So called, because varying the increment INC (in degrees) yields various polygons (try INC=45). The presence of sines and cosines would seem a formidable barrier to translation.

A circle is symmetrical, such that knowing the value of one point, it can be reflected across the x-axis or across the y-axis. That is, if we know (X,Y) is a point on the circumference, then so is (X,–Y). The same is true for (–X,Y) and (–X,–Y). If you divide the circle into quadrants then you only have to code an arc over 0 to 90 deg. These observations also apply to ellipses, (a circle is really a special case of an ellipse) so that we can have a separate x-radius and y-radius.

Potential

The second method is called a 'potential function' and is based upon the realization that the screen cannot plot points except with integers. Every point connects with its neighbor, so if you are at one point on the circumference and trying to figure where the next point is, you can only go in eight directions.

This algorithm is more easily converted to assembly, having as it does only the need to multiply and divide. Note that the slowest routine in Basic can potentially be the fastest in assembly!

Faster CIRCLES

POTENTIAL.PAL is a strict translation of this algorithm which you can relocate where you like. The syntax is:

```
1000 SYS CIRCLE,XC,YC,XR,YR
```

where:

XC is the x axis centre
YC is the y axis centre
XR is the x axis radius
YR is the y axis radius

POLYGON.PAL is a translation of the sines and cosines algorithm reflected about the x and y axes. The syntax is:

```
1000 SYS CIRCLE,XC,YC,XR,YR [,SA,EA,INC]
```

where additionally:

SA is the start angle in degrees
EA is the end angle in degrees
INC is the increment in degrees

Note that the angle parameters are optional and may be omitted in any order, just use commas as place holders.

Source Notes

While the 'potential function' method is fast it is the least flexible and takes up 122 more bytes. The 'polygons routine' gains a lot in speed by using a sine function lookup table. A table of 90 values for 90 degrees yields a resolution of 1 degree. This is admittedly a compromise between table size, and useful resolution. In use, its just as fast and more flexible. Not only do you get circles and ellipses, but also arcs and polygons. And the main loop, which calculates radius and theta, can be readily applied to new graphics commands that require polar coordinates, such as rotation.

A graphic command like CIRCLE, depends so much on fast underlying primitives, like PLOT and DRAW. 'HIRES' has a fast line-DRAW algorithm, but a faster point-PLOT could be coded to make use of a Y-lookup table. Two hundred bytes (0–199 y values) though, would be a lot of extra overhead, it's true!

Coding CIRCLES, gives one a chance to dust off the old computer-math text books or to add new books to your library. Something to play with on cold and rainy winter days.

CIRCLES.BAS: Using the polygon algorithm (170–290) followed by the potential algorithm (310–650). The generator for Garry Kiziak's "HIRES" ML subroutines is not shown here, but can be found in Volume 8 Issue 2 (or on Disk #19). It will also be included on Transactor Disk #21 for this issue.

```
DM  10 rem  'circles'  in basic
LF  20 ifpeek(49152)<>76thenload"hires",8,1
IN  100 :
KE  110 hires = 12*4096: draw = hi + 3: plot = dr + 3
II  120 move = pl + 3: clscr = mo + 3: dmode = cl + 3
GB  130 selpc = dm + 3: colour = se + 3: box = co + 3
LN  140 text = bo + 3: prnt = te + 3: chset = pr + 3
FO  150 trap = ch + 3
EB  160 :
GN  170 rem  basic circles – polygons
CH  180 sys hires,0,1,6
DA  190 xc = 160: yc = 100: xr = 99: yr = 79
IM  200 sa = 0: ea = 360: inc = 5
AG  210 z1 = sa*π/180: z2 = ea*π/180: z3 = inc*π/180
LG  220 x = xc + xr*cos(z1): y = yc + yr*sin(z1)
HO  230 sys move,x,y
DA  240 for i = z1 to z2 stepz3
OJ  250   x = xc + xr*cos(i): y = yc + yr*sin(i)
FA  260   sys draw,x,y
CB  270 next
IL  280 sys prnt,16,12,"polygons"
NP  290 for t = 1 to 2000: next
AK  300 :
HE  310 rem  basic circles – potential
OP  320 sys hires,0,1,6
PI  330 xc = 160: yc = 100: xr = 99: yr = 79
FC  340 phi% = 0: yphi% = 0: xyphi% = 0: y% = 0
        : x% = xr
EJ  350 f = 0: if xr<yr then f = 1
BO  360 if f = 1 then x% = yr: tm = xr: xr = yr: yr = tm
HM  370 rem  start loop
JC  380   yphi% = phi% + y% + y% + 1
BE  390   xyphi% = yphi%–x%–x% + 1
EK  400   if f = 0 then x2 = x%: x3 = y%
FL  410   if f = 1 then y3 = x%: y2 = y%
JK  420   t2% = x%: t1% = t2%*yr/xr
ED  430   if f = 0 then y3 = t1%
ID  440   if f = 1 then x2 = t1%
JM  450   t2% = y%: t1% = t2%*yr/xr
OE  460   if f = 0 then y2 = t1%
KF  470   if f = 1 then x3 = t1%
EF  480   :
EJ  490   sys plot,xc + x2,yc + y2
AK  500   sys plot,xc–x2,yc + y2
OK  510   sys plot,xc–x2,yc–y2
GL  520   sys plot,xc + x2,yc–y2
GI  530   :
MM  540   sys plot,xc + x3,yc + y3
IN  550   sys plot,xc–x3,yc + y3
GO  560   sys plot,xc–x3,yc–y3
OO  570   sys plot,xc + x3,yc–y3
IL  580 :
IB  590   phi% = yphi%: y% = y% + 1
BH  600   if abs(xyphi%)<abs(yphi%) then
          phi% = xyphi%: x% = x%–1
PC  610   if x%>y% goto 380 'loop for more
AO  620 :
MP  630 sys prnt,16,12,"potential"
LF  640 for t = 1 to 2000: next
KI  650 end
```

Creates PRG file POTENTIAL.OBJ

```
FG  1000 rem creates module potential.obj
AC  1010 for j = 1 to 545 : read x
BJ  1020 ch = ch + x : next
NN  1030 if ch<>57836 then print"checksum error"
         : end
FM  1040 print "data ok, now creating file": print
CD  1050 restore
BH  1060 open8,8,8,"0:potential.obj,p,w"
OK  1070 print#8,chr$(0)chr$(128);
GG  1080 for j = 1 to 545 : read x
CM  1090 print#8,chr$(x); : next
ED  1100 close 8
DL  1110 print "prg file 'potential.obj' created. . .
KF  1120 print "this generator no longer needed.
II  1130 rem
GL  1140 data  99,   0,  79,   0,  70,   0,  55,   0
EL  1150 data 162,   3, 189,  43, 192, 157,  47, 192
LK  1160 data 202,  16, 247,  96, 162,   3, 189,  43
NL  1170 data 192, 157,   0, 128, 202,  16, 247,  96
CL  1180 data  32, 135, 193,  32,   8, 128,  32, 135
PK  1190 data 193,  32,  20, 128, 169,   0, 133,  38
IO  1200 data 133,  87, 133,  88, 133,  41, 133,  42
CK  1210 data 173,   0, 128, 133,  39, 205,   2, 128
JK  1220 data 173,   1, 128, 133,  40, 237,   3, 128
IE  1230 data 176,  34, 169, 255, 133,  38, 173,   2
PH  1240 data 128, 133,  39, 170, 173,   3, 128, 133
HP  1250 data  40, 168, 173,   0, 128, 141,   2, 128
NO  1260 data 142,   0, 128, 173,   1, 128, 141,   3
EA  1270 data 128, 140,   1, 128, 166,  42, 134,  90
JF  1280 data 165,  41,  10,  38,  90,  56, 101,  87
```

```
NF  1290 data 133,  89, 165,  90, 101,  88, 133,  90
EC  1300 data 166,  40, 134,  92, 165,  39,  10,  38
AE  1310 data  92, 133,  91,  24, 165,  89, 229,  91
CH  1320 data 133,  91, 165,  90, 229,  92, 133,  92
OG  1330 data 165,  39, 166,  40, 164,  38,  48,  37
MM  1340 data 141,  43, 192, 142,  44, 192,  32, 191
MF  1350 data 129, 141,   6, 128, 142,   7, 128, 165
HH  1360 data  41, 166,  42, 141,   4, 128, 142,   5
KJ  1370 data 128,  32, 191, 129, 141,  45, 192, 142
AJ  1380 data  46, 192,  76, 231, 128, 141,   6, 128
OG  1390 data 142,   7, 128,  32, 191, 129, 141,  43
LD  1400 data 192, 142,  44, 192, 165,  41, 166,  42
AC  1410 data 141,  45, 192, 142,  46, 192,  32, 191
LF  1420 data 129, 141,   4, 128, 142,   5, 128,  32
FO  1430 data  77, 129, 173,   4, 128, 174,   5, 128
EN  1440 data 141,  43, 192, 142,  44, 192, 173,   6
HB  1450 data 128, 174,   7, 128, 141,  45, 192, 142
CO  1460 data  46, 192,  32,  77, 129, 230,  41, 208
OK  1470 data   2, 230,  42, 165,  89, 166,  90, 133
AB  1480 data  87, 134,  88,  32, 175, 129, 133,  36
HC  1490 data 134,  37, 165,  91, 166,  92,  32, 175
PG  1500 data 129, 133,  34, 134,  35, 165,  34, 197
FD  1510 data  36, 165,  35, 229,  37, 176,  16, 165
OE  1520 data  91, 166,  92, 133,  87, 134,  88, 165
CG  1530 data  39, 208,   2, 198,  40, 198,  39, 165
IF  1540 data  39, 197,  41, 165,  40, 229,  42, 144
LD  1550 data   3,  76, 108, 128,  96, 173,  47, 192
EE  1560 data  24, 109,  43, 192, 141,  39, 192,  72
KN  1570 data 173,  48, 192, 109,  44, 192, 141,  40
EI  1580 data 192,  72, 173,  49, 192,  24, 109,  45
EI  1590 data 192, 141,  41, 192, 173,  50, 192, 109
FN  1600 data  46, 192, 141,  42, 192,  32, 117, 195
FP  1610 data 173,  47, 192,  56, 237,  43, 192, 141
KB  1620 data  39, 192, 173,  48, 192, 237,  44, 192
NO  1630 data 141,  40, 192,  32, 117, 195, 173,  49
OA  1640 data 192,  56, 237,  45, 192, 141,  41, 192
KB  1650 data 173,  50, 192, 237,  46, 192, 141,  42
HN  1660 data 192,  32, 117, 195, 104, 141,  40, 192
ID  1670 data 104, 141,  39, 192,  76, 117, 195,  16
CK  1680 data  13,  24,  73, 255, 105,   1,  72, 138
JN  1690 data  73, 255, 105,   0, 170, 104,  96, 133
PL  1700 data  36, 134,  37, 169,   0, 133,  34, 133
FJ  1710 data  35, 162,  17,  24, 102,  35, 102,  34
EN  1720 data 102,  37, 102,  36, 144,  15,  24, 173
KL  1730 data   2, 128, 101,  34, 133,  34, 173,   3
KN  1740 data 128, 101,  35, 133,  35, 202, 208, 228
IA  1750 data 173,   0, 128,  13,   1, 128, 240,  46
LL  1760 data 169,   0, 133,  34, 133,  35, 162,  16
OE  1770 data  38,  36,  38,  37,  38,  34,  38,  35
OE  1780 data  56, 165,  34, 237,   0, 128, 168, 165
DE  1790 data  35, 237,   1, 128, 144,   4, 132,  34
AG  1800 data 133,  35, 202, 208, 227,  38,  36,  38
PH  1810 data  37, 165,  36, 166,  37,  96,  76, 138
MM  1820 data 187
```

Demonstration using POTENTIAL.OBJ

```
FM  10 rem ml circles using potential algorithm
LF  20 if peek(49152)<>76 then load"hires",8,1
OP  30 if peek(32800)<>32 then load"potential.obj",8,1
IN  100 :
KE  110 hires=12*4096: draw=hi+3: plot=dr+3
II  120 move=pl+3: clscr=mo+3: dmode=cl+3
GB  130 selpc=dm+3: colour=se+3: box=co+3
LN  140 text=bo+3: prnt=te+3: chset=pr+3
FO  150 trap=ch+3
EB  160 :
MG  170 circle=32768+32 :rem not the same as
        polygon
CH  180 sys hires,0,1,6
CD  190 :
PC  200 sys prnt,17,1,"circles"
GE  210 :
OC  220 xc=155: yc=100: xr=99: yr=79
EP  230 sys circle,xc,yc,xr,yr
JB  240 sys prnt,19,12,"1"
OG  250 :
HA  260 xc=100: yc=120: xr=90: yr=50
MB  270 sys circle,xc,yc,xr,yr
JJ  280 sys prnt,10,9,"2"
GJ  290 :
MD  300 xc=275: yc=100: xr=30: yr=80
EE  310 sys circle,xc,yc,xr,yr
CH  320 sys prnt,34,12,"3"
OL  330 :
NL  340 xc=52: yc=45: xr=35: yr=30
MG  350 sys circle,xc,yc,xr,yr
HA  360 sys prnt,6,19,"4"
GO  370 :
MD  380 for j=1 to 3000: next
GI  390 end
```

Creates PRG file POLYGON.OBJ

```
JG  1000 rem creates module polygon.obj
DB  1010 for j=1 to 423 : read x
BJ  1020 ch=ch+x : next
AO  1030 if ch<>48466 then print"checksum error"
         : end
FM  1040 print "data ok, now creating file": print
CD  1050 restore
LK  1060 open8,8,8,"0:polygon.obj,p,w"
OK  1070 print#8,chr$(0)chr$(128);
JF  1080 for j=1 to 423 : read x
CM  1090 print#8,chr$(x); : next
ED  1100 close 8
BN  1110 print "prg file 'polygon.obj' created. . .
KF  1120 print "this generator no longer needed.
II  1130 rem
```

```
EE  1140 data   0,   0,   0,   0,   0,   0, 104,   1
LF  1150 data   5,  72,  32, 121,   0, 240,  11,  32
PL  1160 data 253, 174, 201,  44, 240,   4, 104,  76
GO  1170 data 124, 193, 104,  96, 162,   3, 189,  43
OG  1180 data 192, 157,  47, 192, 202,  16, 247,  96
IJ  1190 data 162,   3, 189,  43, 192, 157,   0, 128
ON  1200 data 202,  16, 247,  96,  32, 135, 193,  32
KO  1210 data  28, 128,  32, 135, 193,  32,  40, 128
ID  1220 data 169,   0, 162,   0,  32,   9, 128, 141
MN  1230 data   4, 128, 142,   5, 128, 169, 104, 162
EH  1240 data   1,  32,   9, 128, 141,   6, 128, 142
HH  1250 data   7, 128, 169,   5,  32,   9, 128, 170
IB  1260 data 208,   2, 169,   1, 141,   8, 128, 169
NA  1270 data   0, 133,  91, 133,  92, 173,   4, 128
NF  1280 data 174,   5, 128, 160, 255, 200,  56, 233
FK  1290 data  90, 176, 250, 202,  16, 247, 105,  90
OD  1300 data 133,  87, 152,  74, 144,   7, 169,  90
OF  1310 data  56, 229,  87, 133,  87, 152,  74,  74
PF  1320 data 106, 133,  88, 152,  41,   3, 240,   3
JI  1330 data  56, 233,   3, 133,  89, 173,   2, 128
AF  1340 data 174,   3, 128,  32,  33, 129, 164,  88
AG  1350 data  32,   8, 129,  24, 109,  49, 192, 141
LA  1360 data  45, 192, 138, 109,  50, 192, 141,  46
IE  1370 data 192, 173,   0, 128, 174,   1, 128,  32
OP  1380 data  24, 129, 164,  89,  32,   8, 129,  24
AL  1390 data 109,  47, 192, 141,  43, 192, 138, 109
MB  1400 data  48, 192, 141,  44, 192, 166,  91, 240
LE  1410 data   8,  32,  43, 196, 166,  92, 240,   6
HL  1420 data  96, 198,  91,  32, 113, 194, 173,   8
DK  1430 data 128,  24, 109,   4, 128, 141,   4, 128
JM  1440 data 144,   3, 238,   5, 128, 173,   4, 128
IM  1450 data 205,   6, 128, 173,   5, 128, 237,   7
HN  1460 data 128, 144,   2, 198,  92,  76, 109, 128
AE  1470 data  16,  13,  24,  73, 255, 105,   1,  72
MA  1480 data 138,  73, 255, 105,   0, 170, 104,  96
ND  1490 data  72, 169,  90,  56, 229,  87, 168, 104
JB  1500 data  44, 164,  87, 134,  21, 190,  76, 129
KN  1510 data 134,  34, 133,  20, 169,   0, 133,  35
MP  1520 data 162,   8,  70,  34, 144,  11,  24, 101
FM  1530 data  20,  72, 165,  35, 101,  21, 133,  35
LO  1540 data 104,  70,  35, 106, 202, 208, 235, 133
CH  1550 data  34, 166,  35,  96,   0,   4,   9,  13
OE  1560 data  18,  22,  27,  31,  36,  40,  44,  49
HJ  1570 data  53,  58,  62,  66,  71,  75,  79,  83
FK  1580 data  88,  92,  96, 100, 104, 108, 112, 116
PD  1590 data 120, 124, 128, 132, 136, 139, 143, 147
CK  1600 data 150, 154, 158, 161, 165, 168, 171, 175
CA  1610 data 178, 181, 184, 187, 190, 193, 196, 199
KC  1620 data 202, 204, 207, 210, 212, 215, 217, 219
MG  1630 data 222, 224, 226, 228, 230, 232, 234, 236
GK  1640 data 237, 239, 241, 242, 243, 245, 246, 247
HM  1650 data 248, 249, 250, 251, 252, 253, 254, 254
LO  1660 data 255, 255, 255, 255, 255, 255, 255
```

Demonstration using POLYGON.OBJ. Notice that most of this demo is also in the previous demo. Watch the CIRCLE address in line 170 – it is not the same for both demos. The first page of this article shows the first two sample screens of this program.

```
JO  10 rem ml circles, polygons, arcs
LF  20 if peek(49152)<>76 then load"hires",8,1
AA  30 if peek(32820)<>32 then load"polygon.obj",8,1
IN  100 :
KE  110 hires = 12*4096: draw = hi + 3: plot = dr + 3
II  120 move = pl + 3: clscr = mo + 3: dmode = cl + 3
GB  130 selpc = dm + 3: colour = se + 3: box = co + 3
LN  140 text = bo + 3: prnt = te + 3: chset = pr + 3
FO  150 trap = ch + 3
EB  160 :
MN  170 circle = 32768 + 52 :rem not the same as
            potential
CH  180 sys hires,0,1,6
CD  190 :
OC  200 sys prnt,16,1,"circles"
CN  210 sa = 0: ea = 360: inc = 5
LJ  220 gosub 370
KF  230 :
FP  240 sys prnt,16,1,"polygons"
HJ  250 sa = 0: ea = 360: inc = 45
DM  260 gosub 370
CI  270 :
KN  280 sys prnt,18,1,"arcs"
CM  290 sa = 90: ea = 270: inc = 5
LO  300 gosub 370
KK  310 :
EB  320 sys prnt,16,1,"more arcs"
ME  330 sa = 0: ea = 180: inc = 5
DB  340 gosub 370
OF  350 end
MN  360 :
EM  370 xc = 155: yc = 100: xr = 99: yr = 79
CB  380 sys circle,xc,yc,xr,yr,sa,ea,inc
PK  390 sys prnt,19,12,"1"
EA  400 :
NJ  410 xc = 100: yc = 120: xr = 90: yr = 50
KD  420 sys circle,xc,yc,xr,yr,sa,ea,inc
PC  430 sys prnt,10,9,"2"
MC  440 :
CN  450 xc = 275: yc = 100: xr = 30: yr = 80
CG  460 sys circle,xc,yc,xr,yr,sa,ea,inc
IA  470 sys prnt,34,12,"3"
EF  480 :
DF  490 xc = 52: yc = 45: xr = 35: yr = 30
KI  500 sys circle,xc,yc,xr,yr,sa,ea,inc
NJ  510 sys prnt,6,19,"4"
MH  520 :
CN  530 for j = 1 to 3000: next
LP  540 sys clscr,1,6
CE  550 return
```

Circles: POTENTIAL.PAL

```
MG  100 rem  'hires' circle – potential
PG  110 rem  source file by anthony bryant
JE  120 sys 700
LP  130 .opt n
CA  140 ;
MA  150 ;
GN  160 ;'hires' variables by g.kiziak
KN  170 x1      =   $c027        ;current position
MP  180 y1      =   $c029
AG  190 x2      =   $c02b        ;new position
OB  200 y2      =   $c02d
GJ  210 xc      =   $c02f        ;circ centre (also box)
GF  220 yc      =   $c031
KI  230 hm      =   $c035        ;hires/multi flag
GG  240 ;
AH  250 ;
BL  260 ;'hires' internal subroutines
PM  270 igeti   =   $c17c        ;internal get integer
IO  280 ieget   =   $c187        ;internal eat & get x,y
IK  290 move    =   $c26e        ;'move' rtn
PB  300 imov    =   $c271        ;internal moveto x1,y1
BK  310 iplt    =   $c375        ;internal plot
IJ  320 idrw    =   $c42b        ;internal drawto
AM  330 ;
FI  340 ;zero page labels
PH  350 t1      =   $22
CJ  360 t2      =   $24
LP  370 flag    =   $26
CH  380 x       =   $27
CI  390 y       =   $29
BK  400 phi     =   $57
MG  410 phiy    =   $59
JK  420 phixy   =   $5b
EC  430 ;
KA  440 *=$8000                  ;545 bytes
ID  450 ;
IB  460 xr      .wor 0           ;x radius
FC  470 yr      .wor 0           ;y radius
FO  480 x3      .wor 0           ;potential y
PO  490 y3      .wor 0           ;potential x
KG  500 ;
JH  510 ;subroutine moveto xc,yc
HH  520 movc    ldx   #3
KB  530         lda   x2,x
EI  540         sta   xc,x
NN  550         dex
LN  560         bpl   movc+2
GC  570         rts
KL  580 ;
EB  590 ;subroutine moveto xr,yr
GN  600 movr    ldx   #3
KG  610         lda   x2,x
CP  620         sta   xr,x
NC  630         dex
HG  640         bpl   movr+2
GH  650         rts
KA  660 ;
OB  670 ;sys circle,xc,yc,xr,yr
PG  680 circle  =     *
ND  690         jsr   ieget
MO  700         jsr   movc       ;moveto xc,yc
BF  710         jsr   ieget
FK  720         jsr   movr       ;moveto xr,yr
FM  730         lda   #0
HE  740         sta   flag
PE  750         sta   phi
CD  760         sta   phi+1
BA  770         sta   y
CP  780         sta   y+1
MI  790 ;
LH  800 cases   lda   xr
KG  810         sta   x          ;x=xr
ON  820         cmp   yr
EG  830         lda   xr+1
NC  840         sta   x+1
JI  850         sbc   yr+1
BH  860         bcs   loop       ;branch if xr>= yr
CC  870 swap    lda   #$ff
DN  880         sta   flag
PN  890         lda   yr
CI  900         sta   x
AB  910         tax              ;x=yr
PL  920         lda   yr+1
HI  930         sta   x+1
KE  940         tay              ;and swap
KB  950         lda   xr
DG  960         sta   yr
```

```
EP  970         stx   xr         ;xr with yr
KP  980         lda   xr+1
DE  990         sta   yr+1
MK  1000        sty   xr+1
IG  1010 ;
EL  1020 loop   =     *          ;main loop start
KA  1030        ldx   y+1
ND  1040        stx   phiy+1
LN  1050        lda   y
DA  1060        asl              ;phiy=phi+y+y+1
KB  1070        rol   phiy+1
JL  1080        sec
HF  1090        adc   phi
BB  1100        sta   phiy
JO  1110        lda   phiy+1
OE  1120        adc   phi+1
LD  1130        sta   phiy+1
HH  1140        ldx   x+1
LN  1150        stx   phixy+1
IE  1160        lda   x
EJ  1170        asl              ;phixy=phiy-x-x+1
IL  1180        rol   phixy+1
AM  1190        sta   phixy
GC  1200        clc
BE  1210        lda   phiy
AL  1220        sbc   phixy
IO  1230        sta   phixy
LG  1240        lda   phiy+1
FL  1250        sbc   phixy+1
NO  1260        sta   phixy+1
MG  1270 ;
AM  1280        lda   x
NA  1290        ldx   x+1
JJ  1300        ldy   flag
KL  1310        bmi   altn
KI  1320        sta   x2
CL  1330        stx   x2+1
CC  1340        jsr   scale
LK  1350        sta   y3
DN  1360        stx   y3+1
LB  1370        lda   y
IG  1380        ldx   y+1
CN  1390        sta   x3
KP  1400        stx   x3+1
IG  1410        jsr   scale
PO  1420        sta   y2
HB  1430        stx   y2+1
BL  1440        jmp   doplt
DB  1450 altn   sta   y3
HD  1460        stx   y3+1
EK  1470        jsr   scale
KC  1480        sta   x2
CF  1490        stx   x2+1
NJ  1500        lda   y
KO  1510        ldx   y+1
DF  1520        sta   y2
LH  1530        stx   y2+1
KO  1540        jsr   scale
CH  1550        sta   x3
KJ  1560        stx   x3+1
IJ  1570 ;
NM  1580 doplt  jsr   plot4
MF  1590        lda   x3
EI  1600        ldx   x3+1
MK  1610        sta   x2
EN  1620        stx   x2+1
FI  1630        lda   y3
NK  1640        ldx   y3+1
FN  1650        sta   y2
NP  1660        stx   y2+1
EL  1670        jsr   plot4
GA  1680 ;
LH  1690        inc   y
CN  1700        bne   j1
DA  1710        inc   y+1        ;y=y+1
AO  1720 j1     lda   phiy
HL  1730        ldx   phiy+1     ;phi=phiy
NC  1740        sta   phi
MG  1750        stx   phi+1
MD  1760 abs1   jsr   absv       ;take abs(phiy)
IE  1770        sta   t2
AH  1780        stx   t2+1
KN  1790        lda   phixy
HC  1800        ldx   phixy+1
HF  1810 abs2   jsr   absv       ;take abs(phixy)
IH  1820        sta   t1
AK  1830        stx   t1+1
GK  1840 ;
LC  1850 doif   lda   t1         ;if abs(phixy)
```

```
CL  1860        cmp   t2         ;< abs(phiy)
OC  1870        lda   t1+1
DE  1880        sbc   t2+1       ;then . . .
NL  1890        bcs   else       ;else . . .
FJ  1900 then   lda   phixy
FJ  1910        ldx   phixy+1
BO  1920        sta   phi
HG  1930        stx   phi+1      ;phi=phixy
EF  1940        lda   x
OM  1950        bne   j2
KE  1960        dec   x+1
ME  1970 j2     dec   x          ;x=x–1
CB  1980 else   lda   x          ;if x>= y
EE  1990        cmp   y          ;then loop
HH  2000        lda   x+1
CJ  2010        sbc   y+1
PM  2020        bcc   stop       ;else stop
AM  2030        jmp   loop
BL  2040 stop   rts
IH  2050 ;
OC  2060 ;subroutine reflect points & plot
IF  2070 plot4  =     *
GG  2080        lda   xc
AK  2090        clc
KE  2100        adc   x2
OJ  2110        sta   x1
EM  2120        pha
KF  2130        lda   xc+1
ED  2140        adc   x2+1
II  2150        sta   x1+1
MO  2160        pha
BM  2170        lda   yc
KP  2180        clc
FK  2190        adc   y2
JP  2200        sta   y1
LK  2210        lda   yc+1
FI  2220        adc   y2+1
JN  2230        sta   y1+1
CL  2240        jsr   iplt
AB  2250        lda   xc
FF  2260        sec
CB  2270        sbc   x2
IE  2280        sta   x1
KP  2290        lda   xc+1
CP  2300        sbc   x2+1
IC  2310        sta   x1+1
CA  2320        jsr   iplt
BG  2330        lda   yc
FK  2340        sec
DG  2350        sbc   y2
JJ  2360        sta   y1
LE  2370        lda   yc+1
DE  2380        sbc   y2+1
JH  2390        sta   y1+1
CF  2400        jsr   iplt
CP  2410        pla
GJ  2420        sta   x1+1
GA  2430        pla
IO  2440        sta   x1
KG  2450        jmp   iplt
CB  2460 ;
GM  2470 ;subroutine absolute value
OL  2480 absv   bpl   abok
AD  2490        clc
MG  2500        eor   #$ff
NK  2510        adc   #1
EF  2520        pha
GJ  2530        txa
EJ  2540        eor   #$ff
DN  2550        adc   #0
LM  2560        tax
CJ  2570        pla
PM  2580 abok   rts
EJ  2590 ;
DP  2600 ;subroutine to scale offset
LJ  2610 scale  =     *          ;t1=t2*yr/xr
KJ  2620        sta   t2
CM  2630        stx   t2+1
LD  2640        lda   #0
GL  2650        sta   t1
CI  2660        sta   t1+1
MF  2670        ldx   #17
FB  2680        clc              ;16 bit integer math
FG  2690 mullp  ror   t1+1
BM  2700        ror   t1
DL  2710        ror   t2+1
GN  2720        ror   t2
DA  2730        bcc   decn1
KC  2740        clc
```

```
DC  2750          lda  yr
IN  2760          adc  t1
OC  2770          sta  t1
DA  2780          lda  yr+1
IL  2790          adc  t1+1
OA  2800          sta  t1+1
FE  2810 decn1dex
BO  2820          bne  mullp
CH  2830          lda  xr
EN  2840          ora  xr+1
IF  2850          beq  error
HB  2860          lda  #0
CJ  2870          sta  t1
OF  2880          sta  t1+1
JA  2890          ldx  #16        ;16 bit integer math
HL  2900 divlp    rol  t2
JJ  2910          rol  t2+1
PN  2920          rol  t1
LK  2930          rol  t1+1
NP  2940          sec
EK  2950          lda  t1
EA  2960          sbc  xr
JG  2970          tay
EI  2980          lda  t1+1
EO  2990          sbc  xr+1
CB  3000          bcc  decn2
OH  3010          sty  t1
KO  3020          sta  t1+1
DC  3030 decn2dex
KL  3040          bne  divlp
DG  3050          rol  t2
PC  3060          rol  t2+1
OB  3070          lda  t2
GE  3080          ldx  t2+1
OP  3090          rts
GH  3100 error    jmp  $bb8a      ;"division by zero"
MJ  3110 ;
MA  3120 .end
```

Circles: POLYGON.PAL

```
CF  100 rem  'hires'  circle – polygon
PG  110 rem  source file by anthony bryant
JE  120 sys 700
LP  130 .opt n
CA  140 ;
MA  150 ;
GN  160 ;"hires" variables by g.kiziak
KN  170 x1      =   $c027   ;current position
MP  180 y1      =   $c029
AG  190 x2      =   $c02b   ;new position
OB  200 y2      =   $c02d
GJ  210 xc      =   $c02f   ;circ centre (also box)
GF  220 yc      =   $c031
KI  230 hm      =   $c035   ;hires/multi flag
GG  240 ;
AH  250 ;
BL  260 ;"hires" internal subroutines
PM  270 igeti   =   $c17c   ;internal get integer
IO  280 ieget   =   $c187   ;internal eat & get x,y
IK  290 move    =   $c26e   ;'move' rtn
PB  300 imov    =   $c271.  ;internal moveto x1,y1
BK  310 iplt    =   $c375   ;internal plot
IJ  320 idrw    =   $c42b   ;internal drawto
AM  330 ;
FI  340 ;zero page labels
FG  350 theta   =   $57     ;the angle (0–90deg)
AF  360 ysign   =   $58     ;dependent on quadrant
KM  370 xsign   =   $59     ; "      "
CP  380 ;
IM  390 *=$8000             ;423 bytes
GA  400 ;
GO  410 xr      .wor 0      ;x radius
DP  420 yr      .wor 0      ;y radius
BM  430 arcst   .wor 0      ;arc start (deg)
ME  440 arcend  .wor 360    ;arc end angl (deg)
KK  450 delta   .byt 5      ;polygon incr (deg)
CE  460 ;
HD  470 ;subroutine get angle (deg) integer
JP  480 ;accuracy to 1 deg (hex 5a=90deg)
BK  490 getan   pha         ;save acc
CP  500         jsr  $0079
NG  510         beq  nomore
LJ  520         jsr  $aefd  ;eat ","
CD  530         cmp #","
HI  540         beq  nomore ;another ","!
CN  550         pla         ;throw away acc
HD  560         jmp  igeti  ;get integer to .a & .x
CJ  570 nomore pla
KB  580         rts          ;result in .a & .x
EM  590 ;
DN  600 ;subroutine moveto xc,yc
BN  610 movc    ldx  #3
EH  620         lda  x2,x
ON  630         sta  xc,x
HD  640         dex
FD  650         bpl  movc+2
AI  660         rts
EB  670 ;
OG  680 ;subroutine moveto xr,yr
AD  690 movr    ldx  #3
EM  700         lda  x2,x
ME  710         sta  xr,x
HI  720         dex
BM  730         bpl  movr+2
AN  740         rts
EG  750 ;
LH  760 ;sys circle,xc,yc,xr,yr[,sa,ea,inc]
JM  770 circle  =    *
HJ  780         jsr  ieget
GE  790         jsr  movc     ;moveto xc,yc
LK  800         jsr  ieget
PP  810         jsr  movr     ;moveto xr,yr
PB  820         lda  #0
LP  830         ldx  #0       ;default arcst
HB  840         jsr  getan    ;get sa (degrees)
GE  850         sta  arcst
DJ  860         stx  arcst+1
IP  870         lda  #<360
LN  880         ldx  #>360    ;default arcend
NC  890         jsr  getan    ;get ea (degrees)
IN  900         sta  arcend
KE  910         stx  arcend+1
IP  920         lda  #5       ;default delta
EF  930         jsr  getan    ;get inc (degrees)
HH  940         tax
MK  950         bne  crc1
FH  960         lda  #1       ;minimum
MG  970 crc1    sta  delta
PL  980         lda  #0
IN  990         sta  $5b
FO  1000         sta  $5c
JL  1010 loop    lda  arcst
FP  1020         ldx  arcst+1
DC  1030         ldy  #$ff
LH  1040 ;find quadrant and angle theta
KF  1050 lp2     iny
FK  1060         sec
EL  1070         sbc  #$5a
DF  1080         bcs  lp2
JP  1090         dex
BP  1100         bpl  lp2      ;.y=quadn (0–3)
OL  1110         adc  #$5a
DA  1120         sta  theta    ;(0–90deg)
BC  1130         tya
NE  1140 lsr
MF  1150         bcc  lp3
OP  1160         lda  #$5a
DB  1170         sec
IF  1180         sbc  theta
AJ  1190         sta  theta
NM  1200 lp3     tya
DJ  1210 lsr
NJ  1220 lsr
CI  1230 ror
CM  1240         sta  ysign
JJ  1250         tya
PH  1260         and  #3
FB  1270         beq  lp4
BI  1280         sec
LA  1290         sbc  #3
MH  1300 lp4     sta  xsign
GK  1310 ;do yr*sin(theta)
NI  1320         lda  yr
FL  1330         ldx  yr+1
FI  1340         jsr  calcsin
CF  1350         ldy  ysign
PE  1360         jsr  absv     ;check y sign
AN  1370         clc
NJ  1380         adc  yc
BN  1390         sta  y2
MC  1400         txa
NH  1410         adc  yc+1
BL  1420         sta  y2+1
JB  1430 ;do xr*cos(theta)
EA  1440         lda  xr
MC  1450         ldx  xr+1
MP  1460         jsr  calccos
JM  1470         ldy  xsign
GM  1480         jsr  absv     ;check x sign
IE  1490         clc
EB  1500         adc  xc
IE  1510         sta  x2
EK  1520         txa
EP  1530         adc  xc+1
IC  1540         sta  x2+1
GC  1550         ldx  $5b
AJ  1560         beq  lp5      ;flag a moveto
GH  1570         jsr  idrw     ;drawto
HE  1580         ldx  $5c
LF  1590         beq  lp6
MC  1600         rts
EA  1610 lp5     dec  $5b      ;cancel flag
IP  1620         jsr  imov     ;moveto
CE  1630 lp6     lda  delta
ON  1640         clc
KB  1650         adc  arcst
AH  1660         sta  arcst
AH  1670         bcc  lp7
NE  1680         inc  arcst+1
KL  1690 lp7     lda  arcst
PP  1700         cmp arcend
LE  1710         lda  arcst+1
KO  1720         sbc  arcend+1
PK  1730         bcc  lp8
LN  1740         dec  $5c      ;cancel flag
PB  1750 lp8     jmp  loop
GF  1760 ;
KA  1770 ;subroutine absolute value
CA  1780 absv    bpl  abok
EH  1790         clc
AL  1800         eor  #$ff
BP  1810         adc  #1
IJ  1820         pha
KN  1830         txa
IN  1840         eor  #$ff
HB  1850         adc  #0
PA  1860         tax
GN  1870         pla
NP  1880 abok    rts           ;result in .a & .x
IN  1890 ;
DF  1900 ;subroutine calculate sine func
PP  1910 calccos pha
GP  1920         lda  #$5a
LA  1930         sec
IB  1940         sbc  theta    ;(90–theta)
NG  1950         tay
AD  1960         pla
GH  1970         .byt $2c
NK  1980 calcsin ldy  theta
BD  1990         stx  $15      ;hibyt
FM  2000         ldx  sine,y
GL  2010 calc    stx  $22
AA  2020         sta  $14      ;lobyt
JN  2030         lda  #0
PL  2040         sta  $23
AN  2050         ldx  #8       ;16bit*fract
OK  2060 cal2    lsr  $22
EO  2070         bcc  cal3
GJ  2080         clc
GK  2090         adc  $14
AL  2100         pha
HM  2110         lda  $23
HM  2120         adc  $15
JB  2130         sta  $23
EO  2140         pla
MA  2150 cal3    lsr  $23
EC  2160 ror
BD  2170         dex
HH  2180         bne  cal2
HA  2190         sta  $22      ;reslo in .a
NH  2200         ldx  $23
DF  2210         rts           ;reshi in .x
CC  2220 ;
KO  2230 sine    =    *  ;table of sines (0–90 deg)
ON  2240 .byt $00,$04,$09,$0d,$12,$16,$1b,$1f
OA  2250 .byt $24,$28,$2c,$31,$35,$3a,$3e,$42
LF  2260 .byt $47,$4b,$4f, $53,$58,$5c,$60,$64
ME  2270 .byt $68,$6c,$70,$74,$78,$7c,$80,$84
NB  2280 .byt $88,$8b,$8f, $93,$96,$9a,$9e,$a1
ML  2290 .byt $a5,$a8,$ab,$af, $b2,$b5,$b8,$bb
GA  2300 .byt $be,$c1,$c4,$c7,$ca,$cc,$cf, $d2
BM  2310 .byt $d4,$d7,$d9,$db,$de,$e0,$e2,$e4
IF  2320 .byt $e6,$e8,$ea,$ec,$ed,$ef, $f1, $f2
MB  2330 .byt $f3, $f5, $f6, $f7, $f8, $f9, $fa, $fb
GK  2340 .byt $fc, $fd, $fe, $fe, $ff, $ff, $ff, $ff
GP  2350 .byt $ff, $ff, $ff
OK  2360 ;
OB  2370 .end
```

Inside C128 CP/M: Supporting More Foreign Disk Formats

Mike Garamszeghy
Toronto, Ontario

. . .you can easily add support for virtually any CP/M disk format. . .

One of the nicest features of C-128 CP/M mode is its ability to read and write "foreign" disk formats when used with a 1571 disk drive. This ability is built right into the operating system, so no special user programming is required to access this feature. In addition, with a bit of knowledge of disk formats, you can easily add support for virtually any CP/M disk format. You can even create custom formats of your own to keep your data away from prying eyes. The secret lies in knowing how the CP/M operating system recognizes different formats.

Disk Organization

In order to understand CP/M disk operations, a brief discussion of disk organization is in order. The user area of a CP/M disk is typically divided up into a number of files. Disk space for files is doled out by the operating system in lumps called "blocks" or "allocation units" of a fixed size, typically 1024 bytes for single sided disks or 2048 bytes for double sided disks. Even if a file contains only 1 byte, an entire allocation unit will be required. Unused space in an allocation unit cannot be used by other files, but can be used by the file to which it is assigned if more space is required for it. (To complicate matters, the allocation unit size does not depend on the physical disk sector size, but fortunately none of this is of concern to the casual user).

Each allocation unit is further divided into 128-byte chunks called "records". The record is the smallest directly addressable part of a file. As a file grows (such as by editing a text file), records are added as required. If all space currently allocated to a file is taken up, another allocation unit is allocated. (8 records can fit into a 1024 byte block, or 16 in a 2048 byte block size). A single directory entry has space for 16 allocation units; if more are required extra directory entries or "extents" are created for the file, each taking up 32 bytes of space in the directory area. The directory entry keeps track of the current record count so that space can be added or deleted as required.

Each allocation unit is composed of one or more physical disk sectors. The translation between logical allocation units and physical sectors is handled by the CP/M system using a "translation table" called the DISK PARAMETER TABLE. This contains the data required to convert logical records and allocation units into physical locations on the disk and is completely transparent to user programs, which see everything in terms of standard allocation units and records. The different physical disk formats obviously require different values for the translation parameters. Once these values are set for a given disk format, the user need not be concerned with what the physical disk format is because all disk operations will be adjusted automatically.

The Disk Parameter Table

Information on the disk formats is stored in an area of the CP/M BIOS (or Basic Input/Output System) known as the "DISK PARAMETER TABLE" or DPT. The absolute location of the DPT depends on which version of the CP/M operating system is being used. Three versions are currently in use on the C-128, differentiated by the date displayed on the CP/M boot up screen. For the "1 Aug 85" version, the DPT is located from $d876 to $da75 in BANK 0. For the "6 Dec 85" and "8 Dec 85" versions, it is located at $d6bd to $d8bc. Table 1 is a hex dump of the DPT. It is difficult to examine this from CP/M mode because it resides in BANK 0, while most programs which allow you to examine memory in CP/M mode work in BANK 1. (In CP/M, BANK 0 is used for the operating system while BANK 1 is generally the user work space or "TRANSIENT PROGRAM AREA" (TPA)). You can examine it easily though by using the C-128's MONITOR command in the following way: Boot up CP/M. When you get the CP/M prompt ("A>"), remove the CP/M boot disk then press the <CONTROL> – <ENTER> key combination. (The <ENTER> key is the one at the extreme right of the keyboard in the numeric keypad. It is NOT the same as the <RETURN> key on the main keyboard). This performs a soft reset and gets you back into C-128 BASIC 7.0 mode. From there you can use MONITOR's M command to display the memory.

As you can see, each entry is 32 bytes long and only 9 of the 16 spaces are actually used. The 7 unused spots at the end of the table can be identified by the word "None" in the disk name field. These spots (or any of the other spots, if you do not wish to keep the disk format that it represents) can be filled with custom values.

Each of the bytes in a disk parameter table entry has a specific meaning. These are outlined below. Many of the parameters are repeated in a number of spots in each DPT entry. This is for convenience sake as different parts of the CP/M operating system use the information in different ways. For clarity, the bytes in each entry are numbered from 0 to 31. Bits are numbered 7 6 5 4 3 2 1 0, with 7 being the high bit and 0 being the low bit.

Byte 0 is the root to the "problem" of recognizing a foreign disk format. It can be called a "media descriptor byte" because it is used to identify the physical format of the disk. It is this byte which gets compared when a disk is first read. If two or more entries in the DPT have identical values here, the disk format selection box will appear on the bottom of the screen asking you to choose the correct format. Although it is perhaps the most important, this byte is also very poorly documented (until now). (The only vague reference to it in C-128 documentation is on page 707 of the C-128 Programmers Reference Guide where it is given in cryptic terms such as "S256*2+(16*2-8)+1)". Each of the bits in the byte serves a specific function:

Bit 7 is flag which indicates whether or not to skip track 0 of the disk during an initialization. If it is set to 0 (the normal value), it means that track 0 of the disk is the same as the rest of the disk. If it is set to 1, track 0 has a different format from the rest of the disk. This may seem strange, but many MFM disks (such as Epson QX-10) are formatted differently, typically in single density, on track 0 than on the other tracks to maintain compatibility with much older disk systems.

Bits 5 and 6 indicate the sector size as follows:

5	6	Sector size
0	0	= 128 bytes
0	1	= 256
1	0	= 512
1	1	= 1024

Bits 1 to 4 give an indication of the number of sectors per track. It is actually the binary representation of:

(#sectors/track) - 4

Some typical values are:

4	3	2	1	sectors/track
0	0	0	1	5 = (1)+4
0	1	0	0	8 = (4)+4
0	1	0	1	9 = (4+1)+4
0	1	1	0	10 = (4+2)+4
1	1	0	0	16 = (8+4)+4
1	1	1	0	18 = (8+4+2)+4

Finally, bit 0 gives the minimum sector number on side 0. If the value is 0, then sectors are numbered starting at 0. If bit 0 is set to 1, the sectors start at 1.

Byte 1 is called the Disk Type Byte. It gives additional data on the logical structure of a disk. This byte is described at the bottom of page 717 of the Commodore 128 Programmers Reference Guide. The bits have the following meaning:

Bit 7 describes the disk type: 1 = MFM or 0 = GCR. This is always set to 1 for foreign disk types.

Bit 6 gives tells how the sectors on side 2 of a double sided disk are numbered:

0 = both sides numbered same
1 = side 2 continues from side 1

For single sided disks, the bit should be set to 0. IBM-8 is an example of bit value set to 0. The sectors on both sides are numbered from 1 to 8. Kaypro IV is an example of the second side continuing from the first. The sectors on the first side are numbered from 0 to 9 while the sectors on the second side are numbered from 10 to 19.

Bits 4 and 5 give the sector size, as outlined above for bits 5 and 6 of byte 0.

Bits 1 to 3 determine the order in which a double sided disk is filled. For a single sided disk, the bits should all be set to 0.

3	2	1	
0	0	0	fill track by track, first side 0 then side 1 of same track then next track
0	0	1	fill track by track, even track #'s on side 0, odd on side 1
0	1	0	fill all side 0 then side 1

Bit 0 is a repeat of byte 0, bit 0 as described above.

Bytes 2 and 3 are not normally used. They represent a pointer to a sector skew table. The MFM formats supported by C-128 CP/M do not use a software skew. Sectors are filled in numerical order on a given track such as 1, 2, 3, etc. In this case, these bytes are set to 0. If a skew table were used, bytes 2 and 3 would point to a table containing the order in which the sectors are filled such as, 1, 4, 7, 2, etc. This table is often located in unused entries in the disk parameter table, but can be located anywhere in BANK 0 RAM. It is difficult to set up without a detailed knowledge of both the disk organization and a CP/M memory map for the specific CP/M version that you are using. Fortunately, it is not normally used except for some really weird disk formats such as OSBORNE OSMOSIS and Cromemco.

Bytes 4 to 20 represent a standard CP/M+ DPT entry as described on page 685 of the C-128 Programmers Reference Guide.

Bytes 4 and 5 are referred to as the SPT. This is the total number of 128 byte records per logical track. It is a 16 bit word coded in low byte/high byte format. The size of the logical track depends on both the physical format of the disk and the order in which it is filled. For example, a single sided disk with 8 sectors per track, each 512 bytes, would have 32 records per track. A double sided disk of the same physical format may have either 32 (if the logical track includes only one side of the disk) or 64 (if the logical track includes both sides of the disk) records per logical track, depending on how the disk was filled (see bits 1 to 3 of the disk type byte).

Byte 6 is called the BSH or block shift factor. It is equivalent to the logarithm in base 2 of the number of 128 byte records in a disk allocation unit or LOG 2 (BLS/128). Values are given below.

Byte 7 is the BLM or block mask. It is equal to 1 less than the number of 128 byte records in an allocation unit or (BLS/128) - 1. Values for BSH and BLM are:

BLS (Allocation unit size)	BSH	BLM
1024	3	7
2048	4	15
4096 *	5	31
8192 *	6	63
16384 *	7	127

Note: Block sizes marked with * are not normally used in floppy disk systems. For disks with capacity of greater than 256 k bytes, a BLS of at least 2048 must be used.

The allocation unit size (BLS) is not a direct entry in the DPT but is calculated by the operating system from the BSH and BLM values.

Byte 8 is the extent mask or EXM. This is equal to the maximum number of 16 k file extents that can be coded into a single directory entry. The maximum values are given below. Values of less than the maximum can be used, but this wastes directory space as some of the available spots in the file allocation table will not be used. A value of 0 must be used for disk systems which run under very old versions of CP/M. (For example, if you want to support a disk format that runs on a CP/M 1.4 machine.)

BLS	Typical EXM	
	DSM < 256	DSM > 255
1024	0	-
2048	1	0
4096	3	1
8192	7	3
16384	15	7

Bytes 9 and 10 are a 16 bit word (DSM) representing the total number of allocation units on the disk including the directory area, but excluding reserved system tracks (if any). The bytes are in low byte/high byte format. DSM can be calculated from:

(net # tracks) * (# sectors/track) / (# sectors/allocation unit)

where (net # tracks) is the total number of tracks on the disk minus the reserved system tracks.

Bytes 11 and 12 (DRM) are the total number of directory entries on the disk minus 1. For single sided floppy disks, the number of directory entries is usually 64 and for double sided disks is 128. Therefore, byte 11 is usually either $3F (63) or $7F (127) while byte 12 is normally 0.

Bytes 13 and 14 (called AL0 and AL1) give the number of blocks which are reserved for directory use. This is an inverted 16 bit number of the following format:

```
          <--- Byte 13 ---><--- Byte 14 --->
     Bit   7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
Position   0 1 2 3 4 5 6 7 8 9 A B C D E F
```

Each position which is set to 1 (starting at position 0) indicates one allocation unit reserved for a directory block. Typically, this results in bits 0 and 1 being set, giving a value of $C0 for byte 13 and 0 for byte 14.

Bytes 15 and 16 are called the directory checksum vector, CKS. The 16 bit value is equal to (DRM + 1)/4. For 64 directory entries, byte 15 has a value of $10 (16), while for 128 directory entries it has a value of $20 (32). In both cases, byte 16 has a value of 0.

Bytes 17 and 18 give the track number of the first directory sector (OFF). This is used for disk partitioning and to skip reserved system tracks at the beginning of the disk. Byte 17 is typically in the range of 1 to 4, while byte 18 is normally 0. To create your own custom disk format which uses all of the disk, set both bytes to 0.

Byte 19 is the physical sector shift factor or PSH. It is equal to:

LOG 2([sector-size] / 128).

Values are given below.

Byte 20 is the physical sector mask or PHM. It is equal to:

[sector-size] / 128 – 1

Physical sector size	PSH	PHM
128 bytes	0	0
256 bytes	1	1
512 bytes	2	3
1024 bytes	3	7

Bytes 21 to 31 are not part of a standard CP/M disk parameter table but are used by the C-128 version of CP/M. Byte 21 is the number of sectors on a physical track. Bytes 22 to 31 are an ASCII text string describing the disk name. This is the name which pops up in the disk selection box at the bottom of the screen when the system cannot decide what the format is. Otherwise, it is not used.

Customizing your DPT

What purpose does all of this detailed technical info serve, you ask? Well by playing with the values, you can get your C-128 CP/M to read and write the disks from your uncle Fred's ancient CP/M machine or you can create your own CP/M disk formats that no one else can read to protect your data. (Dear Diary. . ..)

The easiest way to change the DPT is by editing the CPM+.SYS file with a debugger utility such as SID.COM to load the CPM+.SYS file into memory, then change the individual bytes, and save the modified file. Each time you boot up CP/M in the future, you would have automatic support for these other disk formats. This is identical to the method described in the article "CP/M and the 1581 Disk Drive" in Transactor vol 8, Issue 3 (Nov 87). You could also change the parameters in RAM once the computer has been booted, but remember they are stored in BANK 0 RAM and will require special programming in a common area to do this. Try high RAM location starting at $fe00 for the location of such a program. This area is normally used as a disk buffer and is free for non-disk I/O programs which need common BANK 0/BANK 1 memory. Changing the parameters in RAM is only a temporary measure, while changing the CPM+.SYS file is permanent. Take your pick.

The locations of the seven unused disk parameter table entries in the CPM+.SYS file when using SID are as follows:

1 Aug 85 version	6 Dec or 8 Dec 85 version
$1976	$1efd
$1996	$1f1d
$19b6	$1f7d
$19d6	$1f9d
$1a16	$1fbd
$1a36	$1fdd
$1a56	$205d

The end of file (EOF) for the Aug version is $5d00, while for the Dec versions, it is $6400. These numbers will be needed later when rewriting the file. You should always use a backup work disk when doing these changes because your CPM+.SYS file will be changed permanently. Do not modify your original system disk!!!

The disk parameter table entries begin at the bytes specified above, according to the byte parameters previously described. Select an unused spot then use SID's 'D' command to display the memory at that location. For example, d1976 <return> will display memory starting at $1976. If the display does not vaguely resemble the format of table 1, then stop because you do not have the correct memory area. (Check to see that you have the correct version o CP/M then try again). These locations can be changed using SID's 'S' command to the values for the new formats that you wish to support. Once the file has been changed, it can be saved again using SID's 'W' command (w cpm+.sys,100,(EOF value)<return> where (EOF value) is the one listed above for your CP/M version). For a description of how to use these, see the article mentioned above. In order to use your new format, you must do a cold reset (i.e. push the reset button or <CONTROL>–<ENTER>.

Table 2 contains the values for several disk formats which I have added to my system. The MAXI-71 and MAXI-81 formats are ones which I invented myself that take advantage of the full disk capacity (about 396 k for 1571 and 796 k for 1581). These two formats use 1024 byte sectors, double sided, 5 sectors per track, numbered 0 to 4.

To create disks for these new formats, you will need to use the burst mode format command from C-128 mode. For the 1581, the command is:

```
open 15,8,15
print#15,"u0"+chr$(134)+chr$(3)+chr$(79)+chr$(5)
    +chr$(0)+chr$(229)+chr$(0)
close 15
```

For the 1571, the print#15 command is:

```
print#15,"u0"+chr$(102)+chr$(128)+chr$(0)+chr$(3)
    +chr$(39)+chr$(5)+chr$(0)+chr$(0)+chr$(229)
```

Table 1: The unmodified C-128 CP/M disk parameter table

```
>0D876  39 91 00 00 40 00 04 0F 01 97 00 7F 00 C0 00 20:9...@......._.@.
>0D886  00 02 00 01 01 10 45 70 73 6F 6E 20 51 58 31 30:......Epson QX10
>0D896  CD A1 00 00 50 00 04 0F 01 BD 00 7F 00 C0 00 20:M!..P....=._.@.
>0D8A6  00 02 00 02 03 0A 45 70 73 6F 6E 20 51 58 31 30:......Epson QX10
>0D8B6  49 A5 00 00 20 00 03 07 00 9B 00 3F 00 C0 00 10:I%.. ......?.@..
>0D8C6  00 01 00 02 03 08 20 49 42 4D 2D 38 20 53 53 20:...... IBM-8 SS
>0D8D6  49 A5 00 00 20 00 04 0F 01 9D 00 3F 00 80 00 10:I%.. ......?....
>0D8E6  00 01 00 02 03 08 20 49 42 4D 2D 38 20 44 53 20:...... IBM-8 DS
>0D8F6  4C E2 00 00 28 00 04 0F 01 C4 00 7F 00 C0 00 20:Lb..(....D._.@.
>0D906  00 01 00 02 03 0A 4B 61 79 50 72 6F 20 49 56 20:......KayPro IV
>0D916  4C E0 00 00 28 00 03 07 00 C2 00 7F 00 F0 00 20:L@..(....B._.p.
>0D926  00 01 00 02 03 0A 4B 61 79 50 72 6F 20 49 49 20:......KayPro II
>0D936  63 B1 00 00 28 00 03 07 00 B8 00 3F 00 C0 00 10:c1..(....8.?.@..
>0D946  00 03 00 03 07 05 4F 73 62 6F 72 6E 65 20 44 44:......Osborne DD
>0D956  4B A3 00 00 20 00 04 0F 01 9D 00 3F 00 80 00 10:K#.. ......?....
>0D966  00 01 00 02 03 08 20 20 53 6C 69 63 65 72 20 20:......  Slicer
>0D976  39 91 00 00 40 00 04 0F 01 8F 00 7F 00 C0 00 20:9...@......._.@.
>0D986  00 04 00 01 01 10 45 70 73 6F 6E 20 45 75 72 6F:......Epson Euro
>0D996  FF 80 00 00 50 00 04 0F 01 BD 00 7F 00 C0 00 20:_...P....=._.@.
>0D9A6  00 02 00 02 03 08 20 20 20 4E 6F 6E 65 20 20 20:......   None
>0D9B6  FF 80 00 00 50 00 04 0F 01 BD 00 7F 00 C0 00 20:_...P....=._.@.
>0D9C6  00 02 00 02 03 08 20 20 20 4E 6F 6E 65 20 20 20:......   None
>0D9D6  FF 80 00 00 50 00 04 0F 01 BD 00 7F 00 C0 00 20:_...P....=._.@.
>0D9E6  00 02 00 02 03 08 20 20 20 4E 6F 6E 65 20 20 20:......   None
>0D9F6  FF 80 00 00 50 00 04 0F 01 BD 00 7F 00 C0 00 20:_...P....=._.@.
>0DA06  00 02 00 02 03 08 20 20 20 4E 6F 6E 65 20 20 20:......   None
>0DA16  FF 80 00 00 50 00 04 0F 01 BD 00 7F 00 C0 00 20:_...P....=._.@.
>0DA26  00 02 00 02 03 08 20 20 20 4E 6F 6E 65 20 20 20:......   None
>0DA36  FF 80 00 00 50 00 04 0F 01 BD 00 7F 00 C0 00 20:_...P....=._.@.
>0DA46  00 02 00 02 03 08 20 20 20 4E 6F 6E 65 20 20 20:......   None
>0DA56  FF 80 00 00 50 00 04 0F 01 BD 00 7F 00 C0 00 20:_...P....=._.@.
>0DA66  00 02 00 02 03 08 20 20 20 4E 6F 6E 65 20 20 20:......   None
```

Note: The address range given above is for the "1 Aug 85" version of the CPM+.SYS file.
The "6 Dec 85" and "8 Dec 85" start at address $0D6BD.

Table 2: Disk Parameter Table values for selected new disk formats. (Note: all values are in hex)

Byte#	Disk Format					
	XEROX 16-8 DS	OLYMPIA ETX SS	OLYMPIA EX100 DS	TELEVIDEO DS	MAXI-71 DS	MAXI-81 DS
0	4b	4b	4b	3d	62	62
1	a5	a5	a3	91	b0	b0
2	00	00	00	00	00	00
3	00	00	00	00	00	00
4	24	24	24	48	50	50
5	00	00	00	00	00	00
6	04	03	04	04	04	04
7	0f	07	0f	0f	0f	0f
8	01	00	00	00	01	00
9	ae	9b	ae	bd	c8	90
10	00	00	00	00	00	01
11	3f	3f	7f	7f	7f	7f
12	00	00	00	00	00	00
13	80	c0	c0	c0	c0	c0
14	00	00	00	00	00	00
15	10	20	20	20	20	20
16	00	00	00	00	00	00
17	02	02	02	02	00	00
18	00	00	00	00	00	00
19	02	02	02	01	03	03
20	03	03	03	01	07	07
21	09	09	09	12	05	05
22	58	65	4f	54	4d	4d
23	65	74	2d	65	61	61
24	72	78	45	6c	78	78
25	6f	20	58	65	69	69
26	78	49	20	76	20	20
27	20	49	31	69	37	38
28	31	20	30	64	31	31
29	36	20	30	65	20	20
30	2d	20	20	6f	20	20
31	38	20	20	20	20	20

Byte#	Disk Format					
	NCR-DM DS	Zenith Z90 SS	Zenith Z100 DS	Zenith Z100 SS	TRS-80 IV SS	LOBO max SS
0	49	39	49	49	49	3c
1	a5	91	a3	a1	a1	90
2	00	00	00	00	00	00
3	00	00	00	00	00	00
4	20	20	20	20	20	24
5	00	00	00	00	00	00
6	04	03	04	04	03	03
7	0f	07	0f	0f	07	07
8	01	00	00	00	00	00
9	99	97	9b	97	9b	a5
10	00	00	00	00	00	00
11	7f	7f	ff	7f	3f	3f
12	00	00	00	00	00	00
13	c0	f0	f0	f0	c0	c0
14	00	00	00	00	00	00
15	20	20	40	20	10	10
16	00	00	00	00	00	00
17	03	02	02	02	01	03
18	00	00	00	00	00	00
19	02	01	02	02	02	01
20	03	01	03	03	03	01
21	08	10	08	08	08	12
22	4e	48	48	48	54	4c
23	43	65	5a	5a	52	6f
24	52	61	20	20	53	62
25	20	74	31	31	2d	6f
26	20	68	30	30	49	20
27	20	20	30	30	56	20
28	20	39	20	20	20	20
29	20	30	44	53	20	20
30	20	20	53	53	20	20
31	20	20	20	20	20	20

CP/M 3.0: Plus Redirection and Batch Processing

Adam Herst
Toronto, Ontario

The last in the lineage of CP/M for 8 bit computers, CP/M 3.0 was endowed with a name, CP/M Plus, along with its inherited number. This article deals with one of the pluses of CP/M Plus, input/output (I/O) redirection and batch processing.

We are creatures of habit. Introduced to a computer system with the keyboard as the input source and the screen or printer as the output destination, we accept this configuration as the natural order. It isn't. A more flexible conception of an input source and an output destination is possible. The following example illustrates the benefits.

If you're like me, your collection of programs has grown far beyond your capacity to remember which files are on which disks. Labelling the disks with the contents is an obvious solution. Using the DIR command to obtain a list of the files on each disk, then printing it onto a label is the simplest procedure to follow. But wait – to print the label for the disk you'll have to copy the DIR listing down by hand, enter each list into a text editor, format the file for printing and then print it out. There is a better way.

If the output from the DIR command could be redirected to a disk file instead of the screen, then you wouldn't have to copy down and re-enter the information. If the repetitive text editor commands to format the listing for printing could be redirected from a disk file instead of the keyboard then you wouldn't have to tweak each listing by hand. I/O redirection allows you to do just that.

Redirection is a standard feature in most current operating systems. OS's as varied as AmigaDOS, MS-DOS and UNIX provide redirection capabilities. Redirection is commonly provided as a command line option – the redirection operation is indicated on the command line along with the command it is to act upon. Earlier versions of CP/M included limited redirection as options with specific programs and redirection of screen output to the printer through a toggle on the command line. In contrast, CP/M Plus provides a limited implementation of true I/O redirection in addition to these ad hoc capabilities.

The standard notation to redirect program output to a file is:

```
command > output_file
```

The standard notation to redirect program input from a file is:

```
command < input_file
```

It shouldn't surprise you to learn that CP/M Plus does not use the standard command line notation to implement I/O redirection. Instead, CP/M Plus implements I/O redirection with two utility programs invoked as commands in themselves. These are: PUT and GET.

PUT

PUT is a transient command invoked on the command line. It causes the output of subsequent programs to be redirected to the named disk file. Output destined for the screen and output destined for the printer can be placed in a file with the PUT command. Since PUT is a transient command it must be on a logged-in disk in an accessible user area to be executed.

There are four variations of the syntax for the PUT command. Two are related to screen redirection while the other pair is used for printer redirection. The syntax to redirect the printer output of subsequent programs to a file is:

```
PUT PRINTER OUTPUT TO FILE filename.typ
```

This can be abbreviated to:

```
PUT PRINTER FILE filename
```

(This abbreviation holds for all four variations of PUT and will be used for the remainder of this article.) The syntax to restore the direction of program output to the printer is:

```
PUT PRINTER PRINTER
```

To redirect the screen output of subsequent programs to a disk file the syntax is:

```
PUT CONSOLE FILE filename.typ
```

(CONSOLE, abbreviated to CON for some programs, is a special name CP/M uses to refer to the combination of screen and keyboard. When used in an output context, it refers to the display screen. When used in an input context, it refers to the terminal keyboard.) To restore the direction of program output to the screen, the syntax is:

```
PUT CONSOLE CONSOLE
```

By way of example, the following commands must be issued to redirect the output of the DIR command:

```
PUT CONSOLE FILE dir.txt
```

(Since PUT is a transient command, remember to have the file PUT.COM on a logged-in disk in an accessible user area.) CP/M will respond with the message:

```
Putting console output to file dir.txt
```

and attempt a disk access to create the file DIR.TXT. If a file by that name exists, CP/M will prompt for confirmation to overwrite the existing file. Responding with 'no' aborts the PUT command. When the the command line prompt returns, issuing the command:

```
DIR a:
```

sends the output of the DIR program to the file DIR.TXT as well as to the screen.

At this point, the output of subsequently issued commands also would be redirected to the file DIR.TXT. To direct the output of subsequent commands to the screen and to close the redirection file, the command:

```
PUT CONSOLE CONSOLE
```

should be issued. CP/M will respond with:

```
Putting console output to console
```

and close the output redirection file.

If you had wanted the output of the DIR command to go only to the file DIR.TXT and not to the screen as well, PUT's NO ECHO option could have been used. The syntax to include this and other PUT options is:

```
PUT CONSOLE FILE filename.typ [ option_list ]
```

(Note the square brackets surrounding the option list. These must be included on the command line.) Other options for the PUT command include: ECHO, the default, used to restore echoing after a NO ECHO, FILTER, used to translate control characters in the output stream into readable form, NO FILTER, the default, used to cancel filtering after FILTER, and SYSTEM, used to indicate that redirection should occur immediately after the PUT command and include system output as well as program output.

Another, less obvious use of the PUT command is to create files of 0K length. These files are crucial to the operation of certain programs – typically disk cataloguing programs that require files of 0K length to function as disk labels.

A little history is in order. Earlier versions of CP/M, up to release 2.2, included a resident utility named SAVE. Much like the transient utility of the same name provided with CP/M Plus, the earlier version allowed you to save blocks of memory to disk. One quirk in the operation of this command was the ability to save 0 blocks of memory, thus creating a file 0K in length. This quirk was discovered by a number of CP/M hackers and incorporated into programs that required the presence of 0K files.

The modifications made to SAVE in creating SAVE.COM included the removal of this quirk. Users of CP/M Plus cannot create 0K files in this way. This inability has given rise to the practice of distributing 0K files on the disks with the programs that need them! This isn't necessary. In removing the capability from one command, the programmers at DRI introduced it, knowingly or not, into another. PUT can be used to create a file 0K in length. To create a 0K file, redirect output to a file with the command:

```
PUT CONSOLE FILE filename.typ
```

When the prompt returns, instead of issuing a command, restore the direction of output to the screen with the command:

```
PUT CONSOLE CONSOLE
```

(Interestingly, CP/M responds with:

```
Put complete for file: filename.typ
Putting console output to console
```

Since PUT does not respond with the first line in other cases it appears that this was an addition to handle an exception condition. You can almost smell the paste holding this patch on.) You will be left with a file that is 0K bytes in length.

GET

GET is a transient command invoked on the command line to redirect the input of subsequent programs from a named disk file. Only keyboard input can be redirected using the GET command. The input to the program must have been previously placed in the file. Since GET is a transient command it must be on a logged-in disk in an accessible user area to be executed.

At this point it may be helpful to differentiate between two kinds of input to a program: command input and data input. Practically all programs are designed to accept command input from the keyboard as their default. A database management system is a good example. It manipulates data according to 'command' input. If commands aren't entered, nothing happens. The commands entered cause the DBMS to act on 'data' input. If the data is new data, it must be entered into the database via the keyboard. The GET command allows both command input and data input to be taken from a disk file.

The syntax to redirect program output with the GET command is:

```
GET CONSOLE INPUT FROM FILE filename.typ
```

(Used in this context, related to input, CONSOLE refers to the terminal keyboard. Similar to the PUT command, the words CONSOLE INPUT FROM can be omitted and the syntax abbreviated to:

```
GET FILE filename.typ
```

This abbreviation will be used from now on.) Issuing the GET command causes the command input for the subsequent program to be taken from the named file. It is not necessary to issue a command to restore the direction of input from the keyboard. The effects of the GET command, issued without options, will end in one of two conditions. If the redirected input ends the program, the effects of GET will end with the system prompt. If the program is not terminated, the effects of GET will end with the input redirection file, leaving you in the program. In both cases, other programs or commands in the input redirection file, normally issued at the system prompt, will not be executed.

To execute a series of commands from a disk file, GET's SYSTEM option can be used. The syntax for GET with options is:

```
GET CONSOLE FILE filename.typ [ option_list ]
```

(Note the square brackets surrounding the option_list. These must be included on the command line.) This will cause all subsequent input, both system and program, to be taken from the named file immediately. The direction of input from the keyboard will be restored when the end of the input redirection file is reached or input direction is explicitly restored to the keyboard with the command:

GET CONSOLE CONSOLE

We can use GET to automate the process of formatting the directory listing (put into the file DIR.TXT in our last example) for printing, in combination with the line editor supplied with CP/M Plus, ED. (A summary of ED commands is available through the on-line HELP facilities provided with CP/M Plus. If you are thinking of using ED for your general editing needs, be warned that ED is inadequate for interactive, screen-oriented text editing. It is, however, good for editing an input stream in a repetitive pattern.)

First we must create the file that will contain the predetermined input to ED. Place the following input lines into a file called DIR.GET:

```
#a
b
5k
b
ml3kfDirectory^Z-5k
-b
-3k
b
mfUser^Z-4c0kl-2ci........................................^Z
b
mf^L^Z-2c0l40ci^L^Zl
b
#s^L.^Z.^Z
b
mfUser^Zli........................^L........................^L^Z
b
mf^L^Z-2c0l20cki^L^Z
-b
i^L^L^L^L^Z
b
mf^L^Z-2dl-2dl
b
#s^L.^Z.^Z
b
mlfUser^Z-4ci^L^Z60ci^L^Z
b
#s^L^L^Z^L^Z
b
m$1f^L^Zi^L^Z
e
```

Once you have created the file DIR.GET and have put the directory listing in the file DIR.TXT, issue the command:

```
GET CONSOLE FILE DIR.GET
```

(Remember to have the files PUT.COM and DIR.GET on a logged-in disk in accessible user areas.) CP/M will respond with the message:

```
Getting console input from file dir.get
```

Issuing the command:

```
ED DIR.TXT
```

will invoke ED, causing it to take as its input the command lines in the file DIR.GET. (Remember to have ED.COM on a logged-in disk in accessible user areas.) When the end of the file is reached, input will be restored to the keyboard. The file DIR.TXT will be formatted at 60 characters to a line, suitable for printing on a label with 8 lines per label at 17 pitch (132 characters to a line). The command:

```
PIP LST: = DIR.TXT
```

can be used to send the file to a printer.

You may have noticed a couple of problems with our automation process so far. It's not very flexible – there is no way to account for the variations in line spacing on labels. The listed version expects labels that hold eight lines to a label. More problematic is the presence of control codes in the input redirection file for ED. These have been listed in a symbolic representation requiring two characters: the caret (^) and the appropriate alphabetic character. ED will not accept control codes in this representation. Control codes must be entered as their ASCII value. While some text editors will allow you to insert the literal control code within a body of text, most won't. This makes creating the redirection input file extremely difficult. Finally, we've only automated one part of the process required to produce a label of a disk directory – the formatting. There is no reason why the whole process shouldn't be automated. Fortunately, CP/M Plus provides batch processing capabilities that are functionally identical to, but considerably more sophisticated than the capabilities of GET. These are available through the SUBMIT command.

SUBMIT

SUBMIT is a transient command invoked on the command line to redirect system and program input from a named file. This sounds similar to the capabilities of GET. It is – but SUBMIT approaches the problem from a different perspective. While GET was developed to redirect input for a single program – the classic definition of redirection – SUBMIT was developed to execute a series of programs. GET expects program input as its default while SUBMIT expects system (command line) input as its default. While both commands have evolved to accept both kinds of input, the capabilities of SUBMIT provide greater flexibility in the form of variable substitution and symbolic representation of control codes among other features.

A SUBMIT file comprises a series of commands on separate lines (i.e. followed by a carriage return). Issuing the SUBMIT command causes the execution of the commands in the named submit file. Since SUBMIT is a transient command it must be on a logged-in disk in an accessible user area to be invoked. To invoke SUBMIT, issue the command:

```
SUBMIT filename.sub
```

The effects of the SUBMIT command terminate with the end of the submit file. There are no options for the SUBMIT command. At this point, one of SUBMIT's few functional differences from GET becomes apparent. GET redirects input directly from the input redirection file. SUBMIT redirects input from a temporary file created from the .SUB file. For this reason there must be sufficient free disk space on the current disk. If there is not, the SUBMIT command will abort.

To begin automating the disk labelling process, we can create a submit file to put the output from the DIR command into a file for formatting by ED. Create a text file called DSKLABEL.SUB containing the lines:

```
put console file dir.txt
dir.txt [user = all nopage]
put console
```

Issuing the command:

```
SUBMIT DSKLABEL.SUB
```

will cause each line in the submit file to be echoed on the command line and executed. When the end of the submit file is reached, control returns to the system prompt. The file DIR.TXT will contain the directory listing to be formatted.

As it stands now, this submit file will create a directory listing of the logged-in disk. To list the directory of a non-current drive, we would have to log-in to it before issuing the SUBMIT command. It would be convenient if we could specify the disk which is to have its directory listed on the command line when we issue the SUBMIT command. The parameter passing capability of SUBMIT allows us to do just that.

SUBMIT will accept up to nine parameters on the command line. Each argument replaces its corresponding variable in the submit file. Variables take the form of a dollar sign followed by a digit from one to nine. The first argument issued on the command line replaces every occurrence of the variable '$1' in the submit file, the second argument replaces every occurrence of the variable '$2' and so on up to '$9'. (To include an actual '$' in a submit file, use a '$$'.)

To pass the drive specification of the disk which is to have its directory listed, replace the second line of DSKLABEL.SUB with:

```
dir $1 [user=all nopage]
```

Now when the SUBMIT command is issued it can take the form:

```
SUBMIT DSKLABEL drive_specification
```

where drive_specification is the drive letter followed by a colon (e.g. a:). If the drive specification is omitted on the command line, the variable will be replaced by a null value and the directory of the logged-in disk will be listed.

So far we have automated the series of commands to produce the directory listing to be formatted. To format the listing we don't need to execute a series of commands, but need to execute a single program according to a series of program command input lines. To differentiate between programs and program command input lines in a submit file, program command input lines are preceded by a left angle bracket (<). Add the lines listed for the file DIR.GET to DSKLABEL.SUB, preceding each with a left angle bracket. Precede these command input lines with the command ED. Control codes can be represented symbolically – SUBMIT will translate them into literal form before passing them to ED. (To represent a real caret, a '^^' should be used.)

We can accommodate the variations in lines per label that is likely to be encountered using a second variable in the submit file. (Because the lines per label argument will always be passed to the submit file we will use the variable '$1' to represent it. Consequently, the variable representing the drive specification must be changed to '$2'. Since the drive specification is the last argument on the command line, it can still be omitted to no ill effect.) Finally, we can add a line to the submit file invoking PIP to send the formatted directory listing to the printer.

The complete DSKLABEL.SUB file looks like:

```
 1: erase dir.txt
 2: put console file dir.txt [no echo]
 3: dir $2[user=all nopage]
 4: put console
 5: ed dir.txt
 6: <#a
 7: <b
 8: <5k
 9: <b
10: <ml3kfDirectory^Z-5k
11: <-b
12: <-3k
13: <b
14: <mfUser^Z-4c0kl-2ci........................................^Z
15: <b
16: <mf^L^Z-2c0l40ci^L^Zl
17: <b
18: <#s^L.^Z.^Z
19: <b
20: <mfUser^Zli....................^L....................^L^Z
21: <b
22: <mf^L^Z-2c0l20cki^L^Z
23: <-b
24: <i^L^L^L^L^Z
25: <b
26: <mf^L^Z-2dl-2dl
27: <b
28: <#s^L.^Z.^Z
29: <b
30: <mlfUser^Z-4ci^L^Z60ci^L^Z
31: <b
32: <mf^L^L^Z^L^Z
33: <b
34: <m$1f^L^Zi^L^Z
35: <e
36: pip lst:=dir.txt
```

(The exact number of periods in lines 14 and 20 is not important – there should be at least 40 in line 14 and 20 in each of the strings in line 20. This SUBMIT file generates labels from directories that do not have active time/date stamping. If you use time/date stamping omit lines 15 and 16.)

To invoke DSKLABEL issue the command:

```
SUBMIT DSKLABEL labels_per_line drive_specification
```

Where labels_per_line is an integer, drive_specification is a letter followed by a colon representing a disk drive. To create labels with 8 lines per label on the disk in drive M:, the command:

```
SUBMIT DSKLABEL 8 m:
```

should be issued. (The drive specification argument may be omitted resulting in a listing of the current drive, but the labels per line argument must be provided. Failure to do so will result in the double spacing of the DIR.TXT file.) Before issuing the SUBMIT command, remember to set your printer on-line ready to print at 96 or 132 characters per line.

The redirection and batch processing capabilities of CP/M 3.0 are not ideal. The implementation of redirection in the form of free-standing programs is clumsy and unintuitive, although it does allow for the presence of command line options. The absence of true variables and flow of control operators in the SUBMIT environment restricts its usefulness. Nonetheless, these 'pluses' are a vast improvement over CP/M 3.0's predecessors and, in combination with the standard 'tools' provided in the CP/M environment, offer an opportunity to significantly extend the power and usefulness of the C-128.

Square Roots in Machine Language

Jim Butterfield
Toronto, Ontario

. . .Remember those manual square roots?. . .

There are quite a few ways of doing square roots. Basic does the job via the exponential/logarithm functions. Other methods use sucessive approximation: estimate a first value of the root, and then improve it. The usual formula on this last is:

```
new root = (oldroot + value/oldroot)/2
```

But there is a fast, direct method. It may only be worth the coding effort if you have a program which performs a large number of square root calculations. You might find it interesting to trace through. It's very much like the square roots you used to do manually in school.

Remember those manual square roots? They worked something like this:

-- pair off the digits into sets of two;
-- bring down pairs of digits at a time for a trial "root";
-- take the root so far, times 2, as the "multiplier". . .

. . .but it's easier to show in an example. Let's take the root of 34567. We pair off the digits and try for a root of 3:

```
          3 45 67 (1
    1     1
          -
          2
```

It must be a one, since 2x2 would be 4, and that's greater than 3. (Is this coming back to you now?) Bring down the next two digits, and double the root so far to make a "multiplier" (1 times 2 is 2):

```
          3 45 67 (1
    1     1
          -
    2     2 45
```

For the next digit, we might try 9 times 29, 8 times 28. . . that ones works. It gives 224, which is less than 245. So we subtract and repeat:

```
          3 45 67 (18
    1     1
          -
          2 45
   28     2 24
          ----
   36       21 67
```

Note that the new multiplier is 18 times 2, or 36. 9 times 369 is much too big, so we go through 8 times 368, 7 times 367, 6 times 366 (almost!) and finally settle on 5:

```
          3 45 67 (185
    1     1
          -
   28     2 45
          2 24
          ----
            21 67
  365       18 25
            -----
              3 42
```

There's a hefty remainder; we could continue into fractions, but let's leave the result as is for the moment: the root of 34567 is 185 plus a remainder.

We can do all this in binary, and it becomes much easier. Since each digit will be either 0 or 1, our choice is to subtract (1) or not to subtract (0).

Let's go directly to the method. We'll illustrate it graphically. Place the value in a work area, and put a "remainder" buffer filled with zeros at the high end. Set the root to zero:

```
:  REMAINDER  :  VALUE  :   : ROOT :
:0000000000000:01111001:    :000000:
```

For the sake of the example, we'll put a value of 121 (binary 01111001) into the value. Hopefully, the result will be 11 (binary 1011).

Grab two digits – that means two bits in binary turns. We do a "long shift left" on the REMAINDER/VALUE combination. Since we're going to the next "result" digit, we'll shift the ROOT one place left, too:

```
: REMAINDER  : VALUE  :   : ROOT :
:0000000000001:111001..:   :000000:
```

For a multiplier, we take ROOT*2 (same as the decimal method), which is ROOT shifted by one more position. We add 1 to try a "multiply by 1":

```
: REMAINDER  : VALUE  :   : ROOT :  : MULT :
:0000000000001:111001..:   :000000:  :000001:
```

Comparing, we find that MULT is NOT greater than remainder. So we increment ROOT and subtract MULT from REMAINDER. This gives:

```
: REMAINDER  : VALUE  :   : ROOT :
:0000000000000:111001..:   :000001:
```

On to the next pair of bits. As before, a double left shift to REMAINDER/VALUE and a single shift to ROOT. We'll calculate a new MULT at the same time:

```
: REMAINDER  : VALUE  :   : ROOT :  : MULT :
:0000000000011:1001....:   :000010:  :000101:
```

See how MULT is ROOT times 2 plus 1? Well, MULT is bigger than REMAINDER this time, so we skip to the next step with another shift:

```
: REMAINDER  : VALUE  :   : ROOT :  : MULT :
:0000000001110:01......:   :000100:  :001001:
```

MULT is pretty big, but REMAINDER is bigger, so we do our thing on subtraction:

```
: REMAINDER  : VALUE  :   : ROOT :  : MULT :
:0000000000101:01......:   :000101:  :001001:
```

Here comes the last step ... we're shifting the last two bits of value. If we went beyond this point, we'd be into fractions. Here goes:

```
: REMAINDER  : VALUE  :   : ROOT :  : MULT :
:0000000010101:........:   :001010:  :010101:
```

MULT is equal to REMAINDER, so we subtract and increment ROOT.

```
: REMAINDER  : VALUE  :   : ROOT :
:0000000000000:........:   :001011:
```

We're finished. The ROOT is 1011 (decimal 11) and the remainder is zero (correct!)

It's important to provide the right amount of space for the various numbers such as REMAINDER and MULT. It's more than you may think at first.

For example: suppose we're doing the root of an unsigned two-byte number (the range is from 0 to 65535) It's plain that an integer result will fit into one byte (range 0 to 255) so long as we don't need to round it. How big might the remainder be? Well, the biggest remainder would be for SQR(65535). If you work that out, it's 510, which means that nine bits are needed. But wait! That's the remainder AFTER the last subtraction; before that, the value in REMAINDER was 1019. . . a ten-bitter.

In either case, two bytes are needed in this example, for both REMAINDER and MULT. That means we must do a two-byte comparison, and a two-byte subtraction when needed. Don't overlook those extra bits or you'll get wrong answers.

To demonstrate the method, program ROOTS64 does simple square roots. To keep things at their simplest, the program does integers only and uses only single-byte areas for REMAINDER and MULT. As a result, we'd better keep our values in the range of 0 to 4096. That will allow us to keep REMAINDER and MULT within a single byte.

Once you know the principles, it's easy to expand the method to cover bigger numbers, fractions, or whatever fits your objectives.

```
BC  100 v%=rnd(0):rem .. this must be first
BG  110 data 160,   2, 177,  45, 141, 161,   3, 169
HH  120 data   0, 145,  45, 200, 177,  45, 141, 162
MG  130 data   3, 160,   0, 140, 160,   3, 140, 163
LG  140 data   3,  14, 162,   3,  46, 161,   3,  46
MJ  150 data 160,   3,  14, 162,   3,  46, 161,   3
NM  160 data  46, 160,   3,  14, 163,   3, 173, 163
BG  170 data   3,  10, 170, 232, 142, 164,   3, 173
FN  180 data 160,   3, 205, 164,   3, 144,   9, 238
MN  190 data 163,   3, 237, 164,   3, 141, 160,   3
HN  200 data 200, 192,   8, 208, 204, 160,   3, 173
LD  210 data 163,   3, 145,  45,  96
JI  220 for j=828 to 912
FN  230 read x: t=t+x: poke j,x
CN  240 next j
OJ  250 if t<>8554 then stop
GD  260 rem test run starts here
CI  270 for j=1 to 10
FI  280 v=int(rnd(1)*4096)
LF  290 v%=v
AB  300 sys 828
PC  310 print "the root of";v;"is";v%
CC  320 next j
```

PLACEHOLDER for the Commodore 64

Paul Blair
Holder, Australia

In a very early Transactor, Jim Butterfield presented a very handy routine to cope with a perennial screen handling task. The program was designed to allow you to remember the location on screen where you last printed, so that you could go someplace else to print something else, then return to where you left off.

The sort of tasks that spring to mind for this sort of routine would include error messages or continue prompts. It is considered good design to help an operator by always putting help messages at some fixed place on the screen, and the bottom line is a favourite spot.

So you may print on line 2:

ENTER DATE [MM/DD/YY]:

and have a check built in that MM has to be between 1 and 12. If the operator enters "13", it would be nice to remind he or she that any value outside the 1–12 range cannot be accepted. A quick print to the bottom line "PLEASE ENTER 1–12" would be courteous and helpful.

I recently had to arrange this sort of thing from within a machine code program. So, I needed a short routine to do the work for me. PLACE, the extract from that program, shows how it was achieved.

Because the Commodore 64 is so crammed with useful items, it is easy to overlook the possibility that it may be easier to use a routine set up for us by Commodore than to design our own. Such was the case here, because I had overlooked the PLOT routine written by Commodore to permit a sort of PRINT@ on the screen.

PLOT seemed useful to locate to the screen, but PLOT has two sides to it, a fact I rediscovered when I re-read the reference books, something we should all do from time to time!! PLOT can not only set the cursor anywhere on the screen, but it can also work out where the cursor is at any time. By using PLOT to read the screen and remember what it sees, it is easy to duck off to the bottom line (using PLOT to get there), then recall whence the cursor came and go back there for whatever is to happen next.

The Kernal address for PLOT is $FFF0 (or 65520 in decimal). The Kernal is the index to the location of the actual routine, which lives at $E50A (58634). My habit is to use the Kernal addresses, because they might just stay in the same place in whatever machine comes next, whereas I can bet my wife's last dollar that the actual routine will have moved.

Let's look at my machine code first. To READ the screen with PLOT, it is necessary only to set the carry flag (SEC is the instruction) and call PLOT. The X and Y cursor positions (across and down, if you like) will be in the X and Y registers when PLOT finishes its job. Store these values in a handy place (move them to a protected location) or push them onto the stack, and we now know where we have come from.

Now, load X with the "across" value and Y with the "down" value, and go back to PLOT. Print the message you need, then (in this example) wait for a keypress before continuing on. When the keypress comes, recover the old values, load them in the X and Y registers, and PLOT again. Now you are back to where you left off.

The routine looks like this:

```
FB  100 rem save"@0:place.pal"
NG  110 open 8,8,8,"0:place.obj,p,w
JE  120 sys 700
FK  130 .opt o8
IL  140 ;///////////////////////////////
OM  150 ;//                           //
IF  160 ;// machine code placeholder  //
IC  170 ;// for commodore 64          //
MO  180 ;//                           //
LM  190 ;// routine to hold screen    //
JB  200 ;// place, print message on   //
LH  210 ;// last line, then return    //
CB  220 ;// whence thee came          //
OB  230 ;//                           //
HC  240 ;// may 85      paul blair    //
CD  250 ;//                           //
AD  260 ;///////////////////////////////
EI  270 ;
LB  280 ;// c64 basic 2.0 routines //
IJ  290 ;
EP  300 border   =   $d020     ;exterior colour
FE  310 clean    =   $e9ff     ;erase line in .x
KI  320 chrout   =   $ffd2     ;print a char
PG  330 getin    =   $ffe4     ;get a key
JO  340 plot     =   $fff0     ;screen routine
EN  350 ;
EJ  360 ;// main program //
IO  370 ;
KK  380 *        =   $c000     ;sys49152 calls
MP  390 ;
OF  400 ;// error message flip border //
AB  410 ;
JC  420 ohdear   lda  #$02     ;visual error
IC  430          sta  border
OJ  440          sec           ;get our place
AM  450          jsr  plot     ;by reading screen
OL  460          txa           ;for x, y values
FI  470          pha           ;push them away
BM  480          tya           ;for later use
GG  490          pha
FK  500          clc           ;set new location
```

```
CB  510               ldy  #$0c     ;12 over
MN  520               ldx  #$18     ;24 down
FI  530               jsr  plot     ;put cursor there
MM  540               ldy  #>cermsg ;print message
GF  550               lda  #<cermsg ;maybe add a
LN  560               jsr  primms   ;"tone here too?
IG  570 keypls        jsr  getin    ;wait a key
BB  580               beq  keypls   ;loop if no key
EK  590               ldx  #$18     ;erase message
NE  600               jsr  clean    ;on bottom line
EJ  610               lda  #$0f     ;reset border colour
NL  620               sta  border   ;to normal
OP  630               pla
EP  640               tay           ;recall where you
DP  650               pla           ;were before
PF  660               tax
KD  670               clc           ;and go there
OP  680               jmp  plot     ;(rts)
IC  690 ;
NK  700 ;// print messages //
MD  710 ;
HP  720 primms        sty  $5d      ;point to message
HN  730               sta  $5c
GH  740               ldy  #$00     ;counter
HI  750 primm2        lda  ($5c),y  ;get char
EG  760               beq  primm3   ;if zero, end
PN  770               jsr  chrout   ;print it
MM  780               iny           ;inc index
KL  790               bne  primm2   ;loop back
KI  800 primm3        rts           ;all done
AK  810 ;
AI  820 ;// message //
EL  830 ;
MJ  840 cermsg        .byt $12: .asc" press any key "
LK  850               .byt $92,$00.
ID  860 .end
```

A Basic program to load the code, then give a very simple demonstration would be handy.

```
NE  100 rem: placeholder example
CF  110 rem: first load data into $c000
FJ  120 rem: then show off
FG  130 rem: paul blair 5/85
AA  140 :
ND  150 goto 330
EC  160 m=53280: rt=49152: print chr$(31)
DJ  170 poke m, 15: poke m+1, 15
AH  180 print"[CLR]          test placeholder": sys rt
DK  190 print: print"hello there! ";: gosub 290
KB  200 print"from transactor magazine": gosub 290
PL  210 print: print: print"as you can see, ";: gosub 290
JA  220 print"screen control is easy": gosub 290
LD  230 print: print: print"          now i'm here"
        : gosub 290
GC  240 print: print: print: print: print"now down here"
        : gosub 290
GM  250 print: print"have fun!!";: gosub 290
GB  260 print" have fun!!";: gosub 290
```

```
OL  270 print" have fun!!": gosub 290: end
MI  280 :
JM  290 for delay=1 to 1000: next: sys rt: return
AK  300 :
FF  310 rem: load m/c into $c000
EL  320 :
GJ  330 s=49152: f=49237
BO  340 for i=s to f: read a: cs=cs+a: poke i, a: next
NG  350 if cs<>10286 then print"error": end
PJ  360 clr: goto 160
GO  370 :
GF  380 data 169,   2, 141,  32, 208,  56,  32, 240
GK  390 data 255, 138,  72, 152,  72,  24, 160,  12
MJ  400 data 162,  24,  32, 240, 255, 160, 192, 169
LM  410 data  68,  32,  51, 192,  32, 228, 255, 240
PC  420 data 251, 162,  24,  32, 255, 233, 169,  15
MA  430 data 141,  32, 208, 104, 168, 104, 170,  24
LF  440 data  76, 240, 255, 132,  93, 133,  92, 160
ML  450 data   0, 177,  92, 240,   6,  32, 210, 255
GO  460 data 200, 208, 246,  96,  18,  32,  80,  82
DF  470 data  69,  83,  83,  32,  65,  78,  89,  32
HF  480 data  75,  69,  89,  32, 146,   0
```

All those PRINT statements are there to show that old habits die hard. Why not move around the screen using the same routine, but this time in Basic?

How do I do that in Basic? This will involve a bit of PEEKing and POKEing, but it's not too difficult. The prime locations are 781, 782 and 783 (decimal), which are the "save" locations for the X and Y registers, and the flag register.

You will recall that we have to set and clear the carry flag to read/write from/to the screen. The carry flag is Bit 1 (decimal value 2) in the flag register, so we have to twiddle that bit to arrange our "set" and "clear".

To READ the screen, Lines 160 and 170 return X and Y for you.

Lines 180 and 190 move the cursor, and print the message in Line 200. Lines 220 and 230 respond to the keypress.

```
GM  100 rem: plot routine in basic
BF  110 rem: paul blair 5/85
MO  120 :
KL  130 print chr$(147);: sy=65520
AD  140 a=781: b=782: c=783
AI  150 print"hello there ";
OM  160 poke c, peek(c) or 1: sys sy: rem set carry flag
ML  170 x=peek(a): y=peek(b)
CH  180 poke c, peek(c) and 254: rem clear carry flag
DN  190 poke a,24: poke b,14: sys sy
DG  200 print"press a key";
GL  210 get y$: if y$="" then 210
PB  220 poke a, x: poke b, y: sys sy
MF  230 print"from transactor"
```

There you have it. As the screen tells you, have fun!!

Reviews

Reviews

The Turbo Processor for the C64

65C816-based expansion hardware
with 64K of battery backed-up RAM

from SwissComp Inc.

Review By S. Brown Pulliam

The Turbo Processor is one of the more ambitious add-on boards for the C-64 that I have encountered. Physically, it is just an open circuit board, without a case, that plugs into the Cartridge Port. It will require careful handling and anti-static precautions. It replaces almost the entire C-64 with the exception of I/O functions (keyboard, SID, VIC graphics chip, and disk access). All other functions, that is, the microprocessor chip, all RAM, and most of the ROM are replaced by higher performance alternatives. The most spectacular facet of the upgrade is the microprocessor chip itself, the W65SC816. This is a 16 bit CMOS upgrade of the 6502 family with an enhanced instruction set and a 4 MHz clock. The second major feature of the board is a heretofore unheard of (for Commodore) 64K of battery backed RAM. The ROM chip appears to be a 32K device loaded with two 16K operating system replacements for the C-64 Basic and Kernal. The on-board RAM is maintained by a rechargeable NICAD battery.

The board carries a list price of $189.95 (US). It is not overpriced for the amount of hardware it contains, but we must ask what it will to do to improve the operation of our C-64. SWISSCOMP Inc, the Swiss designers of the board, claim first of all that the 4 MHz chip will allow most programs to run 4 times faster, hence the TURBO name. They admit, however, that there are several functions that will not be speeded up. The disk operations, loading and saving, run at the old Commodore speeds, as also must anything having to do with the VIC chip, since it cannot do graphics faster than a 1 MHz clock rate. In addition, any program that uses the CIA chips must also keep the old 1 MHz speed limit.

The other potential advantage for typical commodore users is the non-volatile RAM. With battery backed RAM, if you turn off the computer, or have a sudden power dropout, your program is still stored where you loaded it. As a designer of some battery backed RAM cartridges, I am in a good position to recognize the advantage of having the entire 64K of RAM protected by a battery.

An advantage the TURBO PROCESSOR is supposed to offer to some programmers is a large number of additional machine language commands and increased addressing capability of the 65C816. However, the additional address lines are not made accessible to external RAM on the model I tested, so the 16 megabyte address space is of academic interest only. Nevertheless, the added 16 bit instructions can certainly improve programming efficiency and further speed up many programmed operations, and is one reason, I'm sure, that this microprocessor was chosen for the Apple IIGs. Unfortunately, programs written with 16 bit instructions will only run on C-64s that are also equipped with TURBO PROCESSOR and will not, in general, be compatible with the Apple. The utility of the expanded instruction set is further compromised by the fact that an assembler that supports these instructions may be hard to come by. Perhaps there is one available in Europe where TURBO was originally introduced. I realize it would have been beyond the scope of the manual to do a tutorial on the 65C816 instruction set, but to not even list the instructions, nor mention anything about how to get out of the 6502 emulation mode TURBO seems to stay in, shows they weren't even trying.

The concept of this expansion card is promising, but there are a number of major problems in making it truly useful to the average user. You start off by having to check the positions of 8 dip switches. Then, some users will notice that when the computer is turned on, the screen message will begin to display a few, or on some C-64s, many random characters. The manual instructs you to adjust two small potentiometers (they take a VERY small screwdriver, and one is hard to get at) while continually pressing and releasing the RESET button. On two of the six C-64s I tried this with, I could not find a combination that would get rid of the random characters. Any program that displayed characters on the screen would be virtually useless on these C-64s. Three C-64s had no random characters, and on one, the adjustment did help. The two C-128s I tried to use (in 64 mode) with TURBO would not run by the normal method of holding the C= key when powering up. If you place one of the DIP switches in the OFF position, turn on the C-128 in 64 mode, then move the DIP switch back to TURBO ON, TURBO did run, but with the same severe random characters that incapacitated two of the C-64s.

Assuming your C-64 has no problem with random screen characters, the next hurdle is to find out whether TURBO will let your program run. Start by assuming any cartridge will create a problem, even if you have an extender card. Some programs have made use of undocumented 6510 op-codes, and these will definitely not run with TURBO. There would be a question about disk copy protected software. I tried a recent version of Pocket Writer II which is heavily copy protected, and it would not load with TURBO. A non-protected program written in Basic, or using machine language with a Basic loader should give no problem.

I thought I would try out the speed improvement by running my trusty Paperclip word processor. This program has an 80 column display mode that is nice for viewing text for its format accuracy, but is annoyingly slow when it is scrolling down the page. How nice a 4 times speedup of that scrolling would be. Alas, it was not to be. I plugged in the dongle and inserted Paperclip into the 1541 drive. The first gotcha was that I couldn't use the Auto-Boot utility in my Brown Box to quick load Paperclip with the TURBO PROCESSOR in control at power up, so I un-plugged the Brown Box, and let Paperclip do its normal one minute load. When I got the READY, typing in RUN just hung up the computer. I found this very surprising, since Paperclip is a very tame program.

The instruction manual warns that some games that use video sprites heavily may have some trouble with TURBO. If so, they suggest soldering a three conductor cable from three terminals on TURBO to pins 6,7,and 8 of the PLA chip inside the C-64. The addition then requires a track cut on TURBO and the soldering of a jumper wire. This is not a trivial modification.

I was impressed when I loaded a monitor program and could turn the computer off, then back on, and could SYS to the preserved utility, no matter where in RAM it begins. One important exception is the Cassette buffer which as usual gets zeroed on reset. A Basic program won't run when the computer is turned back on because the power-up reset initializes the Basic pointers. This opens the question of what advantages of this 64K of battery backed RAM can actually be utilized. TURBO will save a program, but, if you want to run it again, you need some software to restore the start up pointers. Even more software would be necessary if you wished to pick up the program with previously generated strings and variables intact. Different software would be required if you wished to store several programs at once. It is possible that the alternative operating system included in its ROM may have some re-initializing software, but if so, they don't mention it in the documentation.

As you may gather, a non-technical user may find TURBO heavy going. Unfortunately, the instruction manual is poorly written, or I should say, translated. The original, in German I would guess, might have read somewhat better. This version is very skimpy, considering the complexity of the product. The technical person is somewhat put off, at least I am, by the obvious pains SWISSCOMP have taken to obliterate the numbers of many of the logic chips on the board. To be useful, a product like TURBO must be documented in such a way that qualified people can learn enough about it to interconnect it with a variety of hardware and software. The SWISSCOMP people are obviously more worried that someone may copy their circuit. I can assure them, from my experience marketing a product line that also has its greatest appeal to the techie user, a good design is only about 10% of the job. The other 90% is communication with the potential user.

TURBO PROCESSOR is a powerful piece of hardware, but I would guess that it will coexist with only a small fraction of commercial programs. It doesn't seem to allow a 1764 RAM Expansion Unit to run if both are mounted on an extender card. If the TURBO connect DIP switch is in the OFF position, 1764 RAM TEST runs successfully, but as soon as it is in the ON mode, the RAM test fails. The designers have not provided software utilities that enable non-technical users to conveniently take advantage of its power. We are therefore left to wonder who will choose to write software to use its capabilities. As a programmer, I have long held the 6502 in special favour, and have hoped that a 16 bit upgrade would become generally available. The logical place for that to have happened would have been when Commodore designed the C-128. Think where a C-128 that had a 65C816 and could directly address its RAM Expander as 512K of continuous memory could be priced relative to the Apple IIGS! It is sad to contemplate that missed opportunity. The most important question is whether a combination of add-on modules for the C-64 makes enough sense from an engineering standpoint to convince programmers that there is a reasonable expectation of a significant market for its software. TURBO PROCESSOR is wasteful. It makes absolutely no use of the internal processor or of the internal 64K of RAM. I must admit my thinking is coloured by my own products, but I submit that added RAM, whether battery backed cartridges such as mine, or volatile RAM such as the Commodore RAM expanders, should not take away the utility of the existing C-64 RAM.

Given the choice between more speed and more speed combined with lots more storage space, the Commodore 1764 RAM Expansion Unit is, in my opinion, a better engineering solution. The 1764's potentiality for faster programs is twofold. Many existing programs that depend on disk file access will run hundreds of times faster once a RAM DOS is loaded. The other speed advantage of the 1764, for new programs, lies in its DMA capability. We have come to think of machine language programs as blindingly fast. The 1764 can be used to move a large block of bytes from one part of C-64 memory to another eight times faster than machine language! When you consider that a program spends much of its time moving bunches of bytes from one part of memory to another, you can see the potential for speed improvement. It should be at least equal to using the full 65C816 instruction set, and the faster clock rate of TURBO. True, the 1764 is volatile RAM and cannot remain loaded when power is off, so that is a trade off against its 256K storage capacity advantage. With an extender card, the 1764 can coexist with a battery backed RAM cartridge, and some programmers might wish a similar capability for TURBO to handle pointer restoration without detracting from the 64K of RAM space. TURBO won't allow this, and it even seems to force a RAM upper limit of 64K because it is incompatible with the 1764 Expansion Cartridge.

Many creative programmers who used to write for the C-64 have moved on to more powerful computers. I am afraid the TURBO PROCESSOR is not the gadget that will lure them back to the 64.

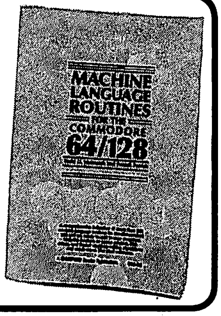
About The Author

Brown Pulliam is Chief Engineer of Brown Boxes, Inc. He designed the QUICK BROWN BOX line of 8, 16, 32 and 64K battery backed RAM cartridges for the C-64 and C-128, and for many years was an Engineer with GENRAD, Inc. He presently consults in the field of electronic testing.

Machine Language Routines for the Commodore 64/128

by Todd D. Heimarck
and Patrick Parrish

from COMPUTE! Publications



Review by Miklos Garamszeghy

Have you ever thought that you have been re–inventing the wheel every time you write an assembly language program? Would you like a large library of documented assembly language source code routines? Or, are you merely curious about how assembly language is actually used in practical applications?

If you answered yes to any or all of the above questions, then you should consider acquiring a copy of "Machine Language Routines for the Commodore 64/128" from COMPUTE! Books. The 580 pages of this recent release contain the assembly language source code for some 200 fully documented, commented, and tested routines for performing a wide range of tasks from file input/output to floating point math to programming the CIA TOD clocks to alphabetizing and searching lists to ASCII <> PETSCII conversions, to name but a few. Although they are not intended to be used as stand alone programs, many can be used with little or no additional programming by incorporating the additional coding provided with each routine to demonstrate its use.

Now that I have tweaked your interest, let me start at the beginning. The first few chapters of the book are devoted to a description of the 6502/6510/8502 type mnemonics instructions such as LDA, STA, etc. and their associated op–codes, and the KERNAL function calls such as CLRCHN, SETLFS, etc. The mnemonic descriptions are similar in style to those found in other COMPUTE! books on machine language programming. The descriptions are short but clear and to the point. References are made in the introduction to several other more comprehensive works on machine language and assembly language programming, including some non–COMPUTE! publications. The descriptions of the KERNAL function calls, which include both standard ones as well as the new C-128 entries, are also short but clear. No real examples of how to use either the KERNAL or the op–codes are given at this point, but all are used extensively in the remainder of the book.

The meat of the book is some 500 pages of carefully explained and documented assembly language routines arranged in alphabetical order by an arbitrarily assigned program name such as ALARM2 (for setting up TOD clock#2 as an alarm clock) or BORCOL (for changing the border color). The routines are presented in a format suitable for a PAL or BUDDY type assembler, but the introduction gives tips and guidance for converting the listings to other assembler formats. If you prefer to do it the hard way, the routines can also be entered using a monitor such as BASIC 7.0's MONITOR command. However, this approach does require much more manual labour on your part for calculating absolute addresses and label references. Tedious, but possible.

The routines are presented in true assembly language format with address and data labels. (Despite the reference to machine language in the title, the book actually deals with assembly language. There is a small but definite difference between the two. Strictly speaking, machine language deals with numbers only while assembly language deals with a set of arbitrarily defined mnemonics used to represent the numbers. For example, $AD $FF $8D $00 $D0 in machine language is equivalent to LDA #$FF : STA $D000 in assembly language). Unlike an earlier COMPUTE! book with a similar title ("Machine Language Routines for the Commodore 64"), BASIC loader DATA statements are not provided for the routines. If you want to SYS to any of them from BASIC, you must create the DATA statements and POKE it in. An assembly language subroutine for creating DATA statements from object code in memory is provided as one of the routines in the book.

Most of the routines are shown as being located in high memory. This follows from the C-64 tradition of putting such things in the unused chunk of RAM at $C000. However, with an assembler, the code is fully relocatable by changing the assembly origin statement. If you want to incorporate more than one of the subroutines or mix them with your own assembly code, relocation is inevitable. It should also be noted that on the C-128 the routines should be placed in low memory, say at $0B00 or $1300, to avoid having to deal with bank switching for accessing the KERNAL and BASIC ROM's as well as the I/O block. Remember that on the C-128, the RAM below $4000 is visible in both BANK 0 and BANK 15 and is an ideal spot for code needing both banks. The introduction discusses these requirements for the C-128 but makes no attempt to explain C-128 bank switching in assembly language (it is very simple) and refers the reader to other sources for an explanation.

Because of the similarity between the two machines, most of the routines will work without modification on either the C-128 or C-64. Where differences exist, due mainly to different addresses for BASIC ROM routines called as subroutines, the documentation for that routine clearly explains the required changes for each machine. Most of the listings are given for the C-64 format with C-128 modifications listed as comments. It should even be possible to adapt the routines to work on other Commodore machines, such as the VIC 20, PLUS/4, etc. Routines designed specifically for one machine or the other, such as accessing the 80 column chip or RAM expander on the C-128, are clearly noted as being such, although I suspect that they would work quite well on a C-128 in C-64 mode with very minor modifications. The RAM expander routines should also work on the C-64 with the 1764 expander.

The collection of routines provided in the book gives something for everyone: from some quite simple routines to very complex ones. A wide range of topics is provided from math and conversion routines to graphics and sound to chip register programming and many more. In fact it covers enough ground to do almost all of the mundane and boring tasks that many programmers don't like to waste a whole lot of time developing for themselves. Face it, most people are lazy, so why reinvent the wheel? If you are really lazy, you can mail in the coupon provided at the end of the book with an extra $12.95 (plus postage, handling and applicable taxes) and get a disk containing the source code for each of the routines in the book in PAL/BUDDY format. Even if you choose not to use the routines exactly as listed, they can serve as a very useful starting point for your own custom routines. A list of some of the routines included is given in Table 1.

Although the introduction states that the book is not intended to be an introduction to assembly language programming, I would recommend this book most strongly for beginner to intermediate programmers. These are the people who would benefit most from the clean programming style and clear documentation of how each routine works. Despite the above statement, the bottom line is that "Machine Language Routines" will be a welcome addition to the library of virtually any assembly language programmer. (Remember, Christmas is not too far away).

TABLE 1:
Selected Assembly Language Routines

Name	Description
ALARM2	Set up CIA TOD alarm clock
ALSWAP	Alphabetize list by swapping strings
ANIMAT	Animation by character sets
BIGMAP	Display a virtual window portion of larger logical screen
CASSCR	Convert PETSCII to screen codes
CHARX4, CHARX8	Print magnified characters
CONCAT	Join two disk files
CUST80	Create custom characters for 80 column screen
EXPLOD	Produce an explosion sound
FETCH/STASH	Retrieve from or store in RAM expander
FIREBT	Read joystick fire buttons
INTCLK	Interrupt driven on screen clock
INTMUS	Interrupt driven background music
MIXLOW	Convert mixed case characters to all lower case
RAS128	Set up a raster interrupt on the C-128 40 col screen
RE80C0/WR80C0	Read or write 80 column chip registers
SPRINT	Interrupt driven sprites
WINDOW	Set C-128 window boundaries
WRBUFF	Open a direct disk buffer a write a disk sector

Also: Many math routines such as addition, subtraction, division of floating point and integer numbers, conversion from ASCII to integer to floating point, etc.

Merlin-128

6502 Macro Assembler Development System

Written by Glen Bredon
Produced by Roger Wagner
Publishing Inc.

Review by M. Garamszeghy

There are a number of good assemblers currently available for the C-128, among them Merlin-128. This entry, written by Glen Bredon and produced by Roger Wagner Publishing Inc. bills itself as a complete macro assembler editor system. Merlin is not copy protected and the publishers even recommend that you make yourself a few backup copies to work with.

When you boot the Merlin disk, you are presented with the main menu from which you can load or save source or object files, access disk utilities, access BASIC's MONITOR, exit to BASIC, etc. The options are all fairly straight forward. Most of the work (except loading and saving files) will be done in the EDITOR/ASSEMBLER mode. The MONITOR can be used to view or test actual assembled object code.

Merlin features an editor with a wide range of screen and line oriented commands for listing, inserting, deleting, moving lines, etc. The commands are entered with combinations of an alpha key with either the control key (for line oriented commands) or the Commodore logo (C=) key (listing oriented commands). Although the commands are fairly logical, they may take some relearning if you are used to another editor, such as Buddy. The Merlin editor assigns line numbers automatically for you in increments of 1. To insert new statements between existing ones, you must manually enter insert mode. Of course, all subsequent line numbers are renumbered automatically. While this feature is handy in some cases, it makes keeping the absolute location of a given source code statement difficult if you move lines around a lot. It also makes it imperative (as stated in the manual) that you delete lines in reverse order. If you delete them in forward order, you will end up deleting the wrong lines due to automatic renumbering after each deletion.

Keyboard macros are supported using the ALT key and function keys. ALT key combinations are used to enter opcodes (e.g. ALT-a will give you LDA, ALT-j gives JSR, etc.) while the function keys will issue editor commands.

The assembler offers a rich vocabulary of pseudo ops and directives. It supports nested macros (up to 15 deep) as well as conditional assemblies, assembly directly from a disk file, multiple source file linking, generation of absolute and relocatable object code and a whole host of other features which most machine language programmers find useful. Although Merlin works primarily with PRG files, SEQ files containing source code can also be accessed from within an assembly by the use of an appropriate PUT pseudo op. Use of SEQ type files allows you to create source code on your favorite word processor, which I find to be a very convenient feature.

Macro libraries are supported with the USE pseudo op. With linked files, labels can be either global or local, with each local label capable of having a different value in different modules. Labels and other values can even be assigned values from the keyboard during assembly.

Text can be handled in the source code in a bewildering variety of ways: as PETSCII text, ASCII text, reverse video, strings with leading length bytes, strings with last character high bit set, etc. The only text format not supported is Commodore screen code. This complicates high speed output to the video chips via direct memory access to video RAM rather than printing to the screen. Numbers can be handled in decimal, hex or binary. Decimal numbers are default. A special pseudo op, FLO, will produce a five byte floating point representation of a number.

The 140 and some page manual, for the most part, is well written and easy to understand. However, considerable confusion is introduced at some points where the text has not been adequately updated from previous versions (see for example the printer command described below). Numerous typographical errors can also be found throughout the text. The manual assumes a certain level of understanding of assembly language concepts and offers no guidance on assembly language opcodes other than a brief description of how Merlin handles certain addressing modes. The reader is referred elsewhere for a tutorial on assembly language. As a plus, a pull out quick reference card is provided. (It would have been nicer to provide it in the form of a keyboard template rather than an alphabetical listing).

The Merlin disk contains a number of example programs, an object file linker and an "unassembler" or source code from object code generator (cutely named "SOURCEROR" in keeping with its mystical image). Each of these utilities is described briefly in the manual.

Although it has some interesting features, Merlin-128 appears to be essentially the C-128 version of an assembler which has been floating around the Apple][world for some time. While this does not mean to say that it isn't any good, it could probably have been made better if it were designed specifically for the C-128 from the ground up. A case in point is certain portions of the instruction manual which were clearly taken from an Apple manual without change for the C-128, even though the context is totally different. For example, the printer command PRTR is given with the syntax: PRTR <slot number>, etc. The slot number refers to the expansion slot in the Apple chassis where the printer interface is located. In the C-128 version, the <slot number> actually refers to the printer device number. The example in the manual uses a <slot number> of 2 (1 in the description of the example, which adds to the confusion) instead of a usual C-128 type device number of 4 or 5.

Because of its Apple heritage, the format of the required assembler source code is perhaps different than what most Commodore users would expect. The format is similar to CP/M and MS-DOS type assemblers with its predefined LABEL, OPCODE, OPERAND, COMMENT fields rather than the free form structure used by PAL, BUDDY and similar assemblers. You are also restricted to one assembly language statement per line. While this makes things uniform, it can lead to overlong listings of simple or standard assembly routines. The source code is definitely not PAL compatible, although it would not take much effort to translate PAL source code. Global search and replace features should make replacing pseudo ops quite easy.

One feature Merlin lacks is a Z-80 cross assembler. Let's face it, the Z-80 on the C-128 is one of its most useful features, yet very few assemblers will support it (Z-BUD from the BUDDY system does an excellent job). A Z-80 cross assembler would allow you to conveniently take advantage of the 16 bit indexing and math modes of the Z-80, something that the 6502 type chip sorely lacks.

In short, if you are looking for a reasonably good C-128 assembler and/or you have used Merlin on another machine, then this program is for you. If you already have a good C-128 assembler, then I can see no reason why you should switch. Merlin is also available for the C-64 and several other 6502 type machines.

For more information, contact Roger Wagner Publishing Inc., 1050 Pioneer Way, Suite P, El Cajon, CA 92020

Benchmark Modula–2

Modula–2 development system
for the Amiga

from Avant–Garde Software

Review by Nick Sullivan

Product : Benchmark Modula-2 Construction Set
Manufacturer : Avant–Garde Software
2213 Woodburn
Plano, Texas 75075
(214)–964–0260

Retail Price	:	Benchmark Modula-2	$199.95
		Simple Libs	$ 99.95
		C Libs	$ 99.95
		Image Resource + IFF Libs	$ 99.95

Update/upgrade policy: Upgrades will be available to registered users of Benchmark Modula-2 for $20–$60 depending on nature of enhancements and amount of new documentation. Bug fixes are not considered upgrades and will be made available at cost of distribution.

The C language is lean, fast and flexible. It gives you the best of both worlds – speed and size that are as close as you can come to pure assembler, plus the data handling and powerful commands of a high level language. Modula–2 is cumbersome by comparison. The code it generates is much less efficient than that produced by a C compiler. Moreover, its strong data typing and Papa–Wirth–knows–best philosophy are ridiculously confining... no wonder it is often described as a voluntary straitjacket for neurotic programmers.

* * * * *

C is perhaps the weakest excuse for a high level language ever developed. The "freedom" it gives you to mix data types is an invitation to disaster, and at best gives you programs that are almost impossible to debug. Its syntax seems to have been specifically designed to produce indecipherable source code. Modula–2 does not have these problems. With M2, most errors are caught at compile time, so you escape the run–time disasters that C programmers all too commonly encounter. The source code is much easier to read, thus much easier to maintain. And the claim that C programs are more "efficient" is just not true – there's no reason at all why a Modula–2 program has to be longer or slower than its C equivalent. If it's maximum speed you want, turn to assembler, not to C... unless you like *making life hard for yourself.*

* * * * *

To date, the contest between C and Modula-2 as the high–level language of choice for Amiga development has been heavily lopsided in favour of C. For one thing, the examples in Amiga programming manuals like the Rom Kernel Manuals are almost exclusively in C, much to the frustration of those who prefer assembler or a different high level language. For another, two excellent C compilers have been available ever since the Amiga came out, whereas for Modula-2 programmers there was only an indifferent offering from TDI. (Note: TDI has recently announced the latest upgrade for their Modula-2 compiler, which is claimed to fix the numerous bugs that have been reported in previous versions.)

With the recent release of Benchmark, a new Modula-2 compiler for the Amiga, the choice of a high level development language is no longer so clear-cut. If the reaction on CompuServe's AmigaForum is any indication, the Benchmark Modula-2 is going to interest a lot of people who want to do serious Amiga programming.

Our review copy of Benchmark arrived at Transactor several weeks ago, which put us immediately in an awkward situation, since none of us knew anything much about the language. That hard fact is the principal background for this article, in which I want to describe the package for you, and tell you about my first foray into M2 programming, which was an M2 translation of the "TWM" C program appearing elsewhere in this issue.

The package first. Our copy came with six heavily loaded disks, and a 2 volume manual totalling more than 700 pages – in other words, an impressive quantity of material. Apart from an extensive set of PD example programs on the disks, mostly adapted from code written for the TDI compiler, all this is the work of one man, Leon Frenkel.

The basic software components of Benchmark (which Frenkel calls a "Modula-2 construction set") are the compiler, linker and editor. These are all invocable separately from the CLI but generally will not be, as both the compiler and linker are also available from function keys within the editor itself. More conveniently still, the editor knows how to read the compiler's error output, which is in a binary format otherwise readable only with the aid of a supplied utility, and lets you step through the offending lines of your source file (again with a function key) to make corrections.

The compiler and linker are fast – even faster than their equivalents in the Manx Aztec C compiler, as far as I can tell – so all in all you have a very quick and friendly environment for developing programs. Executable programs created with Benchmark also run fast – just as fast as the Manx C equivalent, according to Frenkel, though I haven't benchmarked Benchmark and so can't verify that from personal observation. The size of the executable will generally be a bit fatter than Manx would generate, primarily owing (again, according to Frenkel) to the way the Benchmark linker handles libraries – it includes the whole of each referenced library in its output, not just the particular routines used in the program. This has the most noticeable effect on smaller programs (TWM, which is less than 4K under Manx, is about 9K under Benchmark), and may help to account for Benchmark's very fast link times.

The only real disappointment in the Benchmark package, for me, is the editor, which is yet another variant of MicroEmacs – a slick, enhanced Emacs, compared to others I've seen on the Amiga, but Emacs nonetheless. Personally, I find Emacs very close to unusable for heavy editing. However, I found that entering a program in another editor (Rick Stiles' Uedit, in my case), then switching to the supplied editor for compiling, linking and fixing errors was quite satisfactory. It must be admitted, too, that one's choice of text editors is very much a matter of personal taste, so some programmers will undoubtedly be delighted with the inclusion of Emacs in the Benchmark package.

The imposing manual, in our Benchmark, is a special pre–release version, and as such suffers from the haste with which it was obviously prepared – on some pages the typos almost outweigh the text. A great deal of it is given over to documenting the very extensive set of library functions Benchmark provides, and even more is devoted to listings of the ".def" files, which are Modula-2's equivalent of the include files used in C and assembler. Other sections of the manual provide full documentation on the editor (a **lot** of commands, in the true Emacs tradition), the compiler and linker (here you get the usual command line options for specifying input and output directories, search paths for library and symbol files, a switch for including symbol information in the executable file for use with a symbolic debugger like Wack, and so on), and the other supplied utilities. Although the package does not include its own debugger, a full source–level debugger is planned for future release.

One very useful section of the manual provides a set of statement–by–statement guidelines for converting C source to Modula-2, which is a boon for anyone coming to M2, as I did, from a C background, and also for M2 programmers new to the Amiga who are trying to make sense of the heavily C–oriented system manuals. Another section provides detailed instructions for installing Benchmark on a variety of Amiga system configurations, from a one–drive, 512K system on up to a fully loaded Amiga with multiple floppies, hard drive and expansion RAM. It is worth pointing out that, even with the minimum configuration, it is possible to have the editor, compiler and linker resident in RAM during program development, which speeds things up greatly. You may have to play with the sizes of the various internal buffers used by the compiler in order to achieve this, but the necessary steps are fully covered in the manual, and it doesn't sound difficult to do.

My overall impression of the Benchmark so far has been almost uniformly favourable. Frenkel has clearly put a great deal of thought and effort into making the system simple to use for novice programmers without sacrificing power or efficiency. For example, one of the disks that come with Benchmark is a boot disk, configured for a minimum system, that puts you right into the editor. The source for a "Hello World" program is loaded in automatically; you can compile and link it by following the instructions in the comments at the front of the program, and thus get a feeling for using the package within minutes of opening at up, without once looking at the manual. I have never seen a language package for the Amiga that was easier to get started with, AmigaBasic included.

Another indication of the work that has gone into Benchmark is the libraries which, as mentioned above, provide a very rich set of functions beyond the standard Modula-2 set. Some of the libraries are not part of the basic package but may be obtained either by buying the complete system (currently $299 US), or by separate purchase. One of these additional sets of library modules seems to have been specifically designed to seduce diehard C programmers – it contains complete implementations of the standard C libraries for file and terminal input/output, memory allocation, string handling, character conversion and more. Other additional libraries provide functions for simplifying the programmer interface to Intuition, and for handling IFF and other graphics chores in a straightforward way.

As for Modula-2 itself, I admit to mixed feelings. Like its predecessor, Pascal, M2 is the work of Niklaus Wirth, and its flavour derives from Wirth's preoccupation with academic correctness in programming technique. Though M2 is certainly powerful and open–ended enough that you can "beat the system" if you really want to, it does not lend itself to underhanded programming tricks with the same casual ease as C. This can seem confining at first, if you're accustomed to C's free and easy ways, particularly since the two languages are syntactically quite similar.

On the other hand, there are manifest advantages to the comparatively rigid Modula-2 way of doing things. The strict M2 requirement that operands and function parameters be of the correct types eliminates many of the hard to trace bugs that type mismatches introduce into C programs. Even though I find it annoyingly fussy that I can't mix an INTEGER (signed int) with a CARDINAL (unsigned int) in an expression without doing an explicit conversion, I have to grant that it's very nice to pick up potentially disastrous type mismatch problems at compile time.

There are a few Modula-2 features I wish C had counterparts for. One is the SET type, which allows you to treat a collection of objects as an unordered set instead of an ordered array, with appropriate operations for the union and intersection of sets, and for determining whether a particular number is an element of a given set. Another handy feature is the WITH statement, which reduces the overhead for initializing structures (RECORDs, in M2–speak). This not only allows the compiler to generate better object code, but also makes for a cleaner–looking, easier to read source program.

Not surprisingly, there are also features of C that are conspicuously absent from Modula-2. For me, the most aggravating (and inexplicable) is the lack of any way of initializing static data in M2. Where in C you might say something like:

```
int DaysInMonth[] = {31, 28, 31, . . .};
```

which costs you just 24 bytes of object code, since the table is assembled directly into your program, Modula-2 requires you to declare the array separately and initialize it with assignment statements, a comparatively wasteful process. The Benchmark version of the language goes a long way towards dealing with this problem, when dealing with graphics data, by providing a facility for concatenating the data with your program, then accessing it by special purpose functions, but I wish there were a more general solution.

Well, just as editors are matter of taste, so too are languages, and while I'm still dithering over whether Modula-2 is for me, there are undoubtedly many of who have already made up your minds in its favour. If you are one of that many, take a serious look at Benchmark – it's right up there among the best development systems the Amiga has to offer.

TWM in Modula-2

The following listing is a Modula-2 equivalent of the twmClient module given in C elsewhere in this issue. Many of the comments have been omitted to conserve space, but in a few places I have added new comments pointing out differences and similarities between the two languages as reflected in the program. The full Modula-2 source for TWM will appear on the second Transactor Amiga disk, along with the C source, executables, and several programs to which TWM support has been added.

```
(*
  This "definition module" specifies which identifiers in twmClient may be
  accessed by other modules. Except for the main module that every program
  must have, all modules are either "implementation modules" or a
  corresponding "definition module" like this one.
*)

DEFINITION MODULE twmClient;
PROCEDURE PostMe(ClientName: ARRAY OF CHAR) : BOOLEAN;
PROCEDURE UnPostMe();
PROCEDURE twmInit() : BOOLEAN;
PROCEDURE twmCleanUp();
END twmClient.

IMPLEMENTATION MODULE twmClient;

(*
  This module should be compiled and linked with applications that wish to
  be clients of TWM when it is present in the system. Briefly, the client
  calls the function twmInit to set up, afterwards calls PostMe whenever
  he wishes to go to sleep, then finally calls twmCleanUp just before
  exiting. Details are in the prefatory comments to TWM.mod (and TWM.c).

  The FROM statements below correspond roughly to C include statements,
  except that instead of including an entire file, only those identifiers
  that are specifically wanted are brought in (their names appear after
  IMPORT). It is also possible to include ALL the identifiers from a given
  file by saying "IMPORT Memory", for example, but in that case every
  reference to an identifier from that file must be preceded by the name
  of the file plus a period (e.g. "Memory.AllocMem").
*)

FROM SYSTEM IMPORT ADR, BYTE, ADDRESS, TSIZE;
FROM Memory IMPORT AllocMem, FreeMem, MemReqSet, MemClear;
FROM PortsUtil IMPORT CreatePort, DeletePort;
FROM Ports IMPORT
  WaitPort, GetMsg, PutMsg, FindPort, Message, MessagePtr, MsgPortPtr;
FROM Strings IMPORT StringLength;
FROM Nodes  IMPORT NTMessage;

CONST  (* These are constant declarations, similar to C #defines *)

  NULL            = NIL;

  PortName        = "TinyWindowManager";
  GadgNameSize    = 17;

  twmActionAdd    = 0;
  twmActionDelete = 1;
```

```
(* The underscores in the following are not allowed in standard M2, but
  are an extension recognized by the Benchmark compiler *)

  E_OK               =   0;
  E_OPEN_INTUI       = 501;
  E_ALREADY_UP       = 502;
  E_OPEN_PORT        = 503;
  E_OPEN_WINDOW      = 504;
  E_ACTION_UNKNOWN   = 505;
  E_TASK_UNKNOWN     = 506;
  E_NO_MEM           = 507;
  E_ABANDON_SHIP     = 508;


TYPE
  twmMessagePtr  = POINTER TO twmMessage;

(*
  The RECORD (i.e. structure) declaration in the following
  is similar except in syntax details to the C equivalent.
*)

  twmMessage = RECORD
    tmMessage  : Message;
    tmName     : POINTER TO ARRAY [0..GadgNameSize-1] OF CHAR;
    tmAction   : INTEGER;   END;


VAR  (* global variable declarations *)
  mp         : MsgPortPtr;      (* reply port for our msgs                  *)
  twmport    : MsgPortPtr;      (* points to twm's port                     *)
  Addmsg     : twmMessagePtr;   (* twmActionAdd message                     *)
  Delmsg     : twmMessagePtr;   (* twmActionDelete message                  *)
  twmReady : BOOLEAN;           (* TRUE when ports are allocated and initialized *)


(*
  This is a 'function procedure' - it returns a value. The type of the
  returned value is declared after the colon at the end of the first line;
  in this case, it is of type BOOLEAN.
*)

PROCEDURE PostMe (clientName: ARRAY OF CHAR) : BOOLEAN;

VAR  (* declaration of variables local to this procedure *)
  result    : BOOLEAN;

(*
  The variable portGobbler is needed because you are not allowed to ignore
  the value of a function procedure in M2. . . you have to do SOMETHING with
  it. Here, and further on, it is assigned to a meaningless variable.
*)

  portGobbler : ADDRESS;

BEGIN
  result := FALSE;

  (*
    The caret character '^' is used in M2 to dereference pointers. Thus
    Addmsg^.tmName is equivalent to C's Addmsg->tmName or, more precisely,
    (*Addmsg).tmName. Notice that PutMsg and other functions want to be
    RECORDs as arguments, not RECORD pointers. The octothorpe character '#'
    is one of two ways of saying 'not-equal-to' in M2 (the other is '<>').
  *)
  IF StringLength(clientName) # 0 THEN
    IF twmReady THEN
      twmport := FindPort(ADR(PortName));

      IF twmport # NULL THEN
        Addmsg^.tmName  := ADR(clientName);
        Addmsg^.tmAction := twmActionAdd;

        PutMsg(twmport^, Addmsg);

        portGobbler := WaitPort(mp^);

        Addmsg := GetMsg(mp^);

        IF Addmsg^.tmAction = E_OK THEN
          result := TRUE;
        END;
      END;
    END;
  END;
```

```
    RETURN result;
END PostMe;


PROCEDURE UnPostMe;

VAR
  portGobbler   : ADDRESS;
BEGIN

  twmport := FindPort(ADR(PortName));

  IF twmReady AND (twmport # NULL) THEN
    Delmsg^.tmAction := twmActionDelete;
    PutMsg(twmport^, Delmsg);

    (*
      TWM will reply the original (ADD) message before replying this
      one if it's going to reply it at all. . . hence the loop exit condition
    *)

    REPEAT
      portGobbler := WaitPort(mp^);
    UNTIL GetMsg(mp^) = Delmsg;
  END;
END UnPostMe;


PROCEDURE twmInit () : BOOLEAN;

BEGIN

  (*
    Because you can't test for the success of a function like AllocMem()
    in the same statement as you invoke it, a procedure like twmInit is
    significantly bulkier in M2 than in C.
  *)

  IF NOT twmReady THEN
    mp := CreatePort(NULL, 0);

    IF mp # NULL THEN
      Delmsg := AllocMem(TSIZE(twmMessage), MemReqSet{MemClear});

      IF Delmsg # NULL THEN
        Addmsg := AllocMem(TSIZE(twmMessage), MemReqSet{MemClear});

        IF Addmsg # NULL THEN
          Delmsg^.tmMessage.mnReplyPort := mp;
          Addmsg^.tmMessage.mnReplyPort := mp;

          Delmsg^.tmMessage.mnNode.lnType := NTMessage;
          Addmsg^.tmMessage.mnNode.lnType := NTMessage;

          twmReady := TRUE;
        END;
      END;
    END;
  END;

  IF NOT twmReady THEN
    twmCleanUp;
  END;

  RETURN twmReady;

END twmInit;


PROCEDURE twmCleanUp ();

BEGIN
  twmReady := FALSE;

  (* TSIZE in the following is similar to C's sizeof operator. *)

  IF mp       # NULL THEN DeletePort(mp^); END;
  IF Delmsg   # NULL THEN FreeMem(Delmsg, TSIZE(twmMessage)); END;
  IF Addmsg # NULL THEN FreeMem(Addmsg, TSIZE(twmMessage)); END;
END twmCleanUp;

END twmClient.
```

Amiga Dispatches

by Tim Grantham, Toronto, Ontario

It's been almost two years now since I began writing Amiga Dispatches – time to stare into the fire over a foaming tankard, draw reflectively on the pipe and reappraise the future of the Amiga and this column.

This column started as a kind of tip sheet. News was scarce in the conventional media: the only source of current Amiga information was on CompuServe. Most of *that* seemed suspect: "A board that you plug into the 68000 socket to make the Amiga faster than a VAX 11/780? Sure."

Lots of software was coming Real Soon Now. 'Real Soon' turned out to be two years, but it *has* finally arrived – so much so, in fact, that I am very pleased to say that I can't possibly keep up with it anymore.

Those who have been faithful readers of this column have probably noticed a shift in emphasis away from news coverage and towards somewhat more personal commentary. Well, I'm making that official, starting with this edition. From now on, Amiga Dispatches will have a more selective focus than it originally had. It will dispense with breathless announcements of new software and hardware and replace them with more in-depth commentary on a broader range of topics: product trends, programming, computing standards and applications.

I'm sure some of you are curling your lips in disgust at this point. "Oh God, the last thing we need is a Jerry Pournelle clone." I can only say that I will try to keep this column lively, informative and thought-provoking. In that endeavour, I'm fortunate to have the subject I do. Frankly, I still think the Amiga is the most amazing thing since Gandalf slew the Balrog and returned to tell the tale.

All this is not to say that I won't mention the arrival of a game that pushes the hardware to its limits or an expert system that will concoct recipes for benign recreational pharmaceuticals. But such products will have to exemplify what I feel to be genuine innovation.

We at *Transactor* believe that, with the introduction of the 500 and the 2000, the Amiga has a rosy future. The 500 has a very good chance to replace the C64 in the home market over the next five years. It may also become the machine of choice for computer science students: where else can you get a 68020/68881 machine for less than $1800 (US)? Not to mention the multitasking OS and an array of mature yet inexpensive development tools.

The 2000 will continue to find favour with engineers and scientists. I do not believe, however, that the 2000 will make a serious dent in the business market – MS-DOS is too deeply entrenched. But it will be the first Amiga to be taken seriously as a business machine, even more so when it eventually runs Unix.

The 1000 will probably gracefully retire, having spent itself blazing a trail for the 500 and the 2000. It will be brought out on festive occasions and given a place of honour, like restored Bugattis in Canada Day parades.

But even these confident prognostications may fall short of the mark. For I think the Amiga in its various incarnations will provide the platform for genuinely different applications. Given its hardware support for external audio and video signals and the increasing 'digitalization' of television and sound equipment, the Amiga will probably become the first personal computer to be integrated into the average human being's electronic environment.

Meanwhile, it's certainly no slouch in the here-and-now department. Some very exciting work, for example, is being done at the Center for Productivity Enhancement at the University of Lowell. Rich Miner, who is manager of the Center, tells me they are busy porting NCS (Network Computing System) to the Amiga.

Those of you who read my last column may remember NCS: it's Apollo Computers' system for distributed applications that permits a program to run across a network of computers. NCS is network, operating system and hardware independent, and is the first proposal to provide the foundation for truly integrated computing across a heterogeneous environment.

They have many types of machines at the Center: everything from micros to mainframes. And they are all networked together, mostly via Ethernet and NFS (Network File System). Miner (no relation to Jay) sees the Amiga as a a very practical addition to any networked environment. "Why buy a VT-100 terminal when for the same money you can buy an Amiga with an Ethernet interface and be online to several different hosts simultaneously?" When it comes to NCS, Miner believes the Amiga will provide an extremely cost-effective entry into the NCS environment.

Miner is also partly responsible for yet another networking standard being ported to the Amiga: he has given the source code to X-Windows to Dale Luck at Commodore-Amiga.

X-Windows is a graphic interface standard developed at the Massachusetts Institute of Technology and is destined to become the ANSI standard user interface for networked environments. If you have X-Windows running on your machine, it will permit your computer to open a window on any screen on the network, be it an MS-DOS screen, a Mac screen or a Sun workstation screen. Your computer controls that window, wherever it is, exactly as it would one that had opened on its own screen: it can draw in that window and receive input from it.

Like NCS, it has a rival from Sun Microsystems: NeWS, or Network Windowing System. NeWS uses the PostScript page description language developed by Adobe Systems (its founders did the preliminary work on PostScript at Xerox PARC, where all windowing interfaces got their start). With NeWS, each machine on the network has a PostScript interpreter running. The drawing information for each NeWS window is received over the network as a string of PostScript tokens. This means that even a PostScript printer on the network could have a window opened on it and a graphic generated. Of course, such a window could only be used for output.

With an ANSI committee working full steam on X-Windows, though, it looks like it will becomé the official standard. Fortunately for Sun, NeWS can implemented on top of X-Windows. As far as I know, NeWS has not been ported to the Amiga, though it has been made available on the Mac and the Atari ST.

What do NCS and X-Windows mean for the individual? Imagine this: you are an engineer for a giant multinational corporation. Your company has to produce a prototype design of a turbine fan for a high-performance jet engine and it has to do it in *one day*. At 9:00 am, a conference call is set up via satellite with the company's top minds from all over the globe in attendance.

But this is not simply a *voice* conference – all the *computers* that each engineer has access to are also linked up during this conference. Using X-Windows and NCS, the participants can display their design ideas on everyone else's screen; in several different views at once if desired. Various processors on the network could be used to run tests or simulations or analyses on the designs. Participants could modify other designs as well as their own. Once a final design had been settled on, a numerically controlled milling machine somewhere on the network, somewhere in the world, would carve a hunk of metal into the actual prototype.

Of course, such a scenario demands an enormously fast, reliable means of transmitting data. Fibre-optic links can provide such speed now. In fact, according to Cesar Cesaratto, Vice President of Transmission & Hardware Technology for Bell Northern Research, within the decade we will have fibre optics that can transmit 100 terabits per second. With that kind of speed, one half of the world's population could talk to the other half over *two* optical fibres.

Which is ultimately the point – transparent communication between people. In the conference scenario described above, the purpose of X-Windows, NCS, computers, and networks is to provide the invisible support for a free and productive interaction between the participants. And there is no reason why one of those engineers couldn't have been using an Amiga during that conference.

It's this extraordinary range of application of the Amiga – from arcade machines like Bally's **Sub Hunter** to member in good standing in communities of computers – that continues to fascinate me after two years and will, I expect, for many more. Damn, it's a neat machine!

The news. . .

At SIGGRAPH, which is the Association of Computing Machinery's annual conference on computer graphics, the same Center for Productivity Enhancement mentioned above demonstrated a graphics coprocessor board for the Amiga 2000. The **Amiga Parallel Imaging Coprocessor** can use up to seven NEC uPD7281 Image Pipelined Processor chips for an effective processing rate of 35 MIPS. It should be available as you read this and will cost $2000 (US). Full software support, including a library of image processing functions, will be provided with the board. Similar boards for other PCs and workstations can cost $30,000 (US) or more. . .

The Gemstone Group is now selling 68020/68881 boards for the Amiga that plug into the 68000 socket. A fully populated board that includes the 68020 and the 68881 can be had for $775 (US) from The Gemstone Group, 620 Indian Spring Lane, Buffalo Grove, Il 60089. Phone: (312) 537-7405. . . Canadians wishing to obtain **Facc II**, which is a major upgrade to the enhanced disk access program from ASDG Inc., must send in their original disk, and $2.50 in US funds. American owners must send the original disk and a self-addressed, stamped envelope. . . This note from Neil Cumfer in reply to my query in the last edition of *AD*: "**Diga** means 'Speak!' It is what our Spanish-speaking amigos say (instead of 'hello') when they answer the phone." Thanks, Neil. . . In turn, I can enlighten Neil Boyle of Calgary, Alberta who wrote to enquire what 'TANSTAAFL' meant. TANSTAAFL comes from the book *The Moon is a Harsh Mistress* by Robert Heinlein, my personal favourite by that dogged solipsist. It stands for There Ain't No Such Thing As A Free Lunch, and apparently originates in the days of Mr. Heinlein's youth in Missouri when bars would offer free lunch to patrons willing to pay an exorbitant price for the beer. . .

I dropped in on one of the Sunday night conferences on PeopleLink several weeks ago and who should arrive but Rob Peck and RJ Mical! Mical announced that his company, Grab! Inc., had bought the rights to the A-Squared Live! digitizer. Live! had been lost in limbo for some time, as CBM ran hot and cold on it through various corporate convulsions – a case of suspended animation if there ever was one. The device should be available as you read this, for $295 (US).

Speaking of RJ, he and David Needle, another Amiga original, have been hired by David Morse at Epyx, who also hired them when he was CEO of Amiga. They are charged with developing a new line of 'non-software' products for Epyx.

Partners, Inc. here in Toronto have used **Videoscape 3D** to create computer graphics for a Campbell's Soup commercial they have produced for the American market. Look out for it during the World Series. John Foust of *Amazing Computing* tells me they are also using **Sculpt 3D** and his own **Interchanger** to convert object and camera motion files between Sculpt and Videoscape. John is hard at work writing a number of graphics support packages that include a texture-mapping program that lets you project any IFF image onto the surface of a graphic object. You can buy Interchanger for $49.95 (US) from him at Syndesis, 20 West Street, Wilmington, Massachusetts, USA 01887. John is disassociating himself from *Amazing Computing*, partly to avoid conflicts of interest and partly because of dissatisfaction with the way the magazine is being operated. He will continue to act as a technical consultant. . . Sculpt and Videoscape have been joined by animation packages **Forms in Flight** by Micro Magic and **Silver** by Impulse, and by **The Director**, an elaborate slide show program from the Right Answers Group. . . Word Perfect Corp. has established a strong presence on CIS and are actively soliciting bug reports from owners of the Amiga version of **Word Perfect**. They have a good number to work on (of bugs, that is) but are to be commended for their product support. . .

Those of you who just can't find a driver for that printer you've always wanted to hook up can roll your own with the PD program **prtdrvgen** by Joegen Thomsen. It's a lot of work, though. There are some 350 parameters to fill in from your printer's manual. . . Look for **COMAL** soon on the Amiga. . . **Audio Master** from Aegis Development is sound sample editing software that works with all sound digitizing hardware. It costs $59.95 (US). . . SoftCircuits, Inc., who pretty well have the entire computer-aided engineering market for the Amiga to themselves, also have some intriguing PD software available, including a packet radio communications program and a slow scan television program. . . Byte-by-Byte have, after heavy advertising, stopped making their **PAL Jr.** peripheral box. Seems there just wasn't a big enough demand to make it worth while. They will, however, still provide full support for the ones already out there. . .

Some emendations and corrections to last issue's column: I'm glad to see that CBM did post a slight profit in the last quarter and the stock price has recovered somewhat to almost $10 on the NYSE. . . The author of the **Draco** compiler for the Amiga is Chris Gray, not Chris Jeffries. . . SoftCircuits' **Scheme** program sells for $199.95 (US). It is the **Plus** version that sells for $499.95 (US)... My CIS ID number is not 7142*5*,1646 but 7142*6*,1646... Jo-anne Park spells her first name with a hyphen. Sorry, Jo-anne...

Cheath (Charlie Heath, author of **TXeD**) is one of the moderators of BIX's Amiga forums. He and others are working on **arp.library**. This library will contain all the standard C input and output routines, plus other useful routines such as Cheath's **getfile** requester. Programs can then be cut down in size by opening and using arp.library, rather than using code added by the compiler. Of course, the Amiga running the program has to have arp.library in the libs directory on the Workbench disk... Cheath is also the source of an interesting tidbit of programming info: it seems that the Request() function, which opens a requester, can return before the requester has been fully rendered. If an immediate attempt is made to change a gadget in the requester, it may fail. This could be the very reason why I could never get RemoveGadget() and AddGadget() to work in the requester I use in my **keep** program. I eventually gave up, and simply made the changes without removing the gadget concerned first, and then called RefreshGadgets(). It works, if a little crudely. Cheath suggests calling a Delay() function immediately after the Request() function to ensure that the requester has had a chance to be completely rendered in the window..

Transformer 1.2 should be out by the time you read this. Also out is the reason why Commodore hung on to it for so long. Someone posted it to a pirate BBS in the States shortly after it was received at Commodore several months ago. Simile Research, the creators of the program, immediately launched a lawsuit against CBM, blaming them for the leak. According to Simile, apparently, there were only two copies of the program in existence at the time it was handed over to CBM: one was in a safe at Simile's lawyer's office; the other was given to Commodore. With the resolution of the lawsuit, the program has been released. (As a side note, the same programmers behind Transformer have created **PCDitto** for the Atari ST, which provides PC emulation with colour graphics. It apparently cannot do monochrome.)

Speaking of emulators, Randy Linden here in Toronto has accomplished a rather amazing feat of programming. He has written a C64 emulator for the Amiga that can handle such arcana as raster interrupts, fast loaders and code intended for the SID chip. In a recent demonstration at the Transactor offices, Linden's emulator ran every program thrown at it, except for some copy protected products, though not at full speed. Among the programs were several that pump the graphics and sound pretty hard.

Unlike the C64 emulator I have mentioned in previous columns, Linden's does not require a hardware interface – unless you call a cable an interface. The program can use a 1541 connected via the cable to the Amiga's serial port, or a regular 3.5" Amiga drive can be used to partition and format a portion of a disk in 1541 format.

The program is not an absolutely perfect emulation: in addition to the reduced speed (roughly half that of the 64, depending on the program), rapidly changing raster interrupts can cause it to stumble, as well as double-wide sprites. Nor does it work with all fast loaders; because they are so dependent on timing, Linden has to write custom code for each one.

At this point, Linden is thinking of selling the program for $49.95 (Can.), $69.95 with the cable. At last, relative files on the Amiga!

I have been growing rather weary lately of the endless moaning from some quarters about the deficiencies of AmigaDOS or the lack of a cheap, fast hard drive. I have pointed out that AmigaDOS has had nowhere near the time and money put into its development as Unix or MS-DOS. As for hard drives, no-one can produce a cheap one for a market consisting of at most 200,000 machines. The economies of scale just don't come into play.

I was gratified then to receive this response from Chris Siebenmann, currently a programmer at Gold Disk:

> "*Sigh. I guess we just haven't talked recently about the good aspects of the Amiga, like multitasking, and a system architecture that let me build a print spooler with three simple programs, or the fact that the Amiga (with enough memory) is just about my ideal development environment. I know of no other machine on which so many neat hacks (PopCLI, ClickToFront, FACC, vd0:, and many more) could have been done so transparently and work together so well. It's a wonderful system... which is why the bad bits provoke me so much.*
>
> *As to the hardware... amen. When I buy expansion hardware, I'm going to pay extra for ASDG or Comspec reliability without even blinking. It's worth it. For what you get, the prices AREN'T out of line. Name one other machine with cheap autoconfiguring HDs. I'll give you a hint... it isn't made by IBM.*"

Coming up next time...

CBM Canada has kindly lent me a 2000 for evaluation. The full report will be in the next edition of *Amiga Dispatches*, but I can tell you right now, I'm thoroughly spoiled. I don't want to have to go back to my 1000. In addition to the standard 1 Mb of RAM, this 2000 has a 20 Mb hard disk shared by the Amiga and a Bridge card. The hard disk is considerably faster than the floppies: a typical compile and link seems to take only about one quarter the time it did with floppies. An interesting side-effect of having the extra .5 Mb of RAM is that I have started using the Workbench again, rather than just the CLI. It really shows that the Amiga doesn't come into its own until you have a decent amount of memory to work with.

The 2000 on loan to me is a West German model. As you probably know, it will be replaced by the so-called West Chester design, also known as the B2000. This has some significant improvements, including a video slot with a higher number of signals available, a 'Fat Agnes' chip and 1 Mb of RAM on the CHIP bus, which opens the door to a possible upgrade on the graphics performance, and a cleaner design overall. I hope to be able to bring you a review of this machine as well. Hmmm. Maybe I can just get CBM to keep lending me review machines...

By the way, Commodore has just published *The A500/A2000 Technical Reference Guide*, which provides complete schematics for the 500, the West German 2000 and the West Chester 2000, in-depth hardware info and complete Bridgecard documentation, including the Janus library calls. It can be ordered from Lauren Brown (mistakenly referred to as 'Laurie' in my last column – sorry, Lauren) at CBM West Chester in PA, for $40 (US).

Also coming up are reports on the arrival of **TeX** for the Amiga, version 4 of the Lattice C compiler, and on AmiExpo, the Amiga Exposition to be held October 10-12, in New York. I was invited by the organizers to appear on one of the user panels but, to my bitter disappointment, could not raise the funds to go. So I am dragooning Nick and Karl, who *are* going, into grabbing every bit of information they can for me.

Until then, keep those cards and letters, electronic and otherwise, coming, folks. They're appreciated.

CIS: 71426,1646	GEnie: t.grantham
PeopleLink: AMTAG	Bloom Beacon BBS: Tim Grantham
BIX: dispatcher	(416 297-5607)

TWM

A Paneless Approach to Tiny Window Management

by Nick Sullivan

Most programs on the Amiga can be divided into three fairly tidy classes. The commonest class consists of programs like DIR and LIST, that you invoke as commands, that do their work then exit. Another class consists of handlers, like the console handler ConMan, or PopToFront, from a few Transactors ago. These programs, or their offspring, live in the system usually until next reboot but, because they require no user interaction, they are invisible.

Programs in the third class are the ones you interact with for an extended period of time, such as text editors, terminal emulators and paint programs, or that you might keep around for sporadic interaction, such as PopColours and Structure Browser. One benefit of the Amiga's multitasking environment is that you don't have to take such programs down in order to do something else. You can switch readily from your editor to your terminal, for instance, and keep your text in memory; you can switch from the terminal back to the editor and stay on-line.

The extent to which you can take advantage of this capability depends, of course, on how much RAM you have in your system in relation to the size of the programs you're running. Even with a lot of expansion RAM, though, you are still limited by the amount of available "chip RAM" – the special area of memory that the Amiga's custom chips can use. On current Amigas, chip RAM is limited to 512K and, while that sounds like a lot, it can quickly get eaten up by programs that use lots of windows, colourful screens, gadgets, and other display elements that need chip RAM to survive.

The other problem with running a lot of interactive programs simultaneously is that they tend to crowd your monitor screen. That makes for a lot of depth-arranging and resizing as you flit from one task to another – the infamous "electronic shell-game" – and can get pretty tiresome if you have to do a lot of it. A few programs even put up a full-size window and won't allow you to get at the Workbench screen behind.

One approach that some programs have taken to relieve the on-screen congestion has been to supply a "tiny window mode", which can be invoked when the program is not in active use. This idea was arrived at independently quite a while ago in at least two programs I know of – Rick Stiles' shareware text editor Uedit, and Chris Zamara's PopColours. In Uedit particularly, use of the tiny window (invoked by clicking on the editor's title bar) achieves a significant savings in chip RAM. Using a normal 640 by 200 window on the Workbench screen, which has two bit-planes, Uedit needs 32K for its bit-map, plus a bit more for gadgets. Its tiny window, however, is a mere 100 by 20 pixels in size, and so consumes less than 600 bytes. Clearly, the chip RAM penalty for running concurrent applications would be considerably eased if the use of a tiny window mode was more widespread.

A tiny window consists of no more than an inch or two of title bar with an equivalent thickness of empty window beneath. It is draggable, and may be depth arranged (since part of its purpose is to keep the application that owns it out of your hair), but not resizable. Clicking in the empty part reactivates the parent program, prompting it to take the tiny window down, put its working window (or screen) back up, and carry on with business as usual.

One reason for this article is to advocate the use of tiny windows in programs – including commercial programs – in which their use is appropriate (for one approach to implementing a tiny window mode see the listings for "TWM" and "Test1" below). Suppose this idea *were* generally adopted, though, making it easier to run several such programs concurrently. Now the user has another problem: the new disorder of TWL (Tiny Window Litter), in which one's visible workspace is obscured by annoying swarms of tiny windows that continually seem to be getting in your way as you work, no matter how much you try to shuffle things around.

So the other reason for this article is to present TWM, for 'Tiny Window Manager', a small and easily implemented piece of code that enables programs to support a tiny window mode while giving users a method of avoiding the anguish of TWL, and the consequent disruption of their lives.

From the user's point of view, TWM is a kind of central storage compartment in which sleeping programs are housed, and from which they can be activated. The programs do not have to maintain any display of their own – not even a tiny window, so the user's screen is free from clutter. TWM's own working window contains gadgets bearing the names of its "client programs". When the user clicks on one of these gadgets, the corresponding client program is awoken and resumes operation. TWM also has its own tiny window mode; when that is in use, the amount of chip RAM jointly consumed for graphics by the client programs and TWM itself is very small. When the system is hosting two or more applications that support TWM, there is a significant savings in both resources and convenience. Of course, even if TWM is not running, applications that support it will run normally – but instead of disappearing entirely when they go to sleep, they will put up a tiny window in the usual way.

From the programmer's point of view, TWM comes in two parts – the program TWM itself, and a short C-language module called twmClient.c (also available in Modula 2 – see the review of Benchmark Modula 2 in this issue). The twmClient code can be compiled and linked with any application that supports a tiny window mode. Let us suppose that this client application has been running in its active mode but now, as a result of some action of the user's (perhaps a menu selection, perhaps clicking on a gadget) it has taken down its working display and is about to put up its tiny window and go to sleep.

Before taking that step, the application now calls the function PostMe() in the twmClient module, passing as an argument the name by which it would like to be known, as in:

```
PostMe("PopColours");
```

PostMe(), in its turn, searches the system for a public message port with the name "TinyWindowManager". If the search succeeds, PostMe() sends a message to that port with the name of the client, and waits at its own message port for a reply. Effectively, the client application has now let itself go to sleep and, because it has closed its working window, there are no visible signs of its existence.

The message sent by the client is now picked up by the TWM program, which the user has earlier run, and which is now displaying one of its own windows (either the tiny window or the larger working window) on the user's screen. On receipt of the message TWM creates a gadget bearing the client application's name. The gadget will be displayed in TWM's working window (immediately, if that window is up). There may be other gadgets in the window also – one for each client application. This is the only indication that the clients still exist and, when TWM is in its tiny window mode, there is no sign of them at all. Chip RAM is conserved, and the user's window is uncluttered. When the user later clicks on the gadget, TWM replies to the message the client sent, deletes the gadget, then forgets about the client altogether.

Back now to PostMe(), waiting asleep at its message port for a reply to its message. The reply has finally come, signifying that the user has selected the client's gadget in the TWM working window, and wants the client to put up its own working window again. PostMe() now returns to the client, with the value TRUE, and the client goes back to work.

Several things might have gone wrong along the way. The most probable of these is that the user may not currently have TWM running. A remoter possibility is that TWM might have failed to allocate memory for the client's gadget, or could not open a window. In all these cases, PostMe() returns FALSE to the client, who then knows that it is necessary to put up a tiny window of its own after all.

As you will see in the code that follows, there are other details. In case the client application wishes to wake itself up (in response to time-out or some other kind of message) while it is in TWM's care, an UnPostMe() function is also provided. Most clients won't need UnPostMe(); in that case, the programmer can remove UnPostMe() from twmClient.c to shrink the code even further. Another detail is that TWM remembers where the user last placed its windows, and restores them to the chosen position each time they are re-opened. Uedit and PopColours also have this feature, and it is recommended that other tiny window programs include it (see the pertly named SavePosCloseW() function below for sample code).

The intent of TWM is to institute a standard of which all tiny window programs can take advantage. Therefore all the following code is freely redistributable, and may be used in any program – PD, shareware, or commercial. New versions of PopColours and the XE expression evaluation program from Transactor's Amiga Disk #1 now support TWM, and will be available on CompuServe's AmigaForum, along with TWM itself, by the time this magazine is in your hands. They will also appear on our second Amiga Disk, which should be out in January.

LISTING 1: TWM.C

```
/* TWM.c

  Tiny-Window Manager v1.0      (c) 1987 Transactor Publishing Inc.
  by Nick Sullivan

  This program is freely redistributable provided that no charge is made
  for the redistribution beyond reasonable reproduction costs, and that no
  changes are made except with the prior written approval of Transactor
  Publishing Inc., 85 West Wilmot St. #10, Richmond Hill, Ontario L4B 1K7.

  TWM provides a storage area in which applications that are inactive, but
  running, can wait to be re-activated without using any chip RAM. It thus
  provides an alternative to the "tiny window" approach to minimizing chip
  RAM use, as exemplified by such programs as Uedit and PopColours.
    When TWM is run, it puts up its own tiny window, and creates a public
  message port. Client applications should check for the existence of this
  port and, if it is present, send a "twMessage" (as defined in twm.h) with
  the twm_action field set to TWM_ACTION_ADD when they wish to go to sleep.
  When TWM's tiny window is clicked in thereafter, a larger window will be
  put up containing gadgets bearing the names of each client application
  that has been added. Clicking on one of these gadgets will cause the
  twMessage to be replied to, which is the signal for the client to
  reawaken. At the same time as the reply is sent, the large TWM window will
  be taken down, and the gadget for that client removed.
    If a client wishes to reactivate itself before its TWM gadget is
  clicked, or if it wishes to exit altogether, it should first send a
  twMessage with the twm_action field set to TWM_ACTION_DELETE.
    The twm_action field is used by TWM to return a code to the client that
  indicates whether the requested operation was successful. The code for
  success is E_OK. Other possibilities are:

  E_NO_MEM              A client has asked TWM to add a gadget, but TWM
    was unable to allocate memory for the gadget structure.
  E_ABANDON_SHIP        1) A client has asked TWM to add a gadget while
    TWM has its large window up. In this case, TWM closes the large window,
    and re-opens it after rethinking the gadget positions and the window size.
      2) The user has clicked on TWM's tiny window, or closed its large
    window, causing TWM to close the current window and attempt to open the
    other one.
      If the window open fails in either of these cases, TWM sends all
    current clients this error message, then exits, since it has no means
    of recovery.
  E_TASK_UNKNOWN        A client has sent a TWM_ACTION_DELETE, but TWM
    does not currently have a gadget for that client.
  E_ACTION_UNKNOWN      A message has been received with the twm_action field
    set to an unknown code. . . currently the only possibilities are
    TWM_ACTION_DELETE and TWM_ACTION_ADD.

  Sample C code for applications wishing to interface correctly with TWM
  is contained in the file twmClient.c. The header file twm.h is also
  required. Before using other functions in this file, the client should
  call PostMe(), supplying as an argument the name that should appear on its
  gadget in the TWM window. The name can be up to GADGNAMESIZE - 1
  characters in length (currently 16 characters); excess characters are
  removed.
    The function PostMe() in twmClient.c should be called when the
  client wishes to deactivate. If this function returns TRUE, the client
  should resume its life as an active application. If it returns FALSE,
  either TWM is not present in the system or else the attempt to post failed
  for some reason. In this case, the client should take an alternative
  approach to deactivated living (like making its own tiny window), or else
  not allow itself to be deactivated.
    Programs that wish to be able to receive messages even when deactivated
  (time-outs, for example), will need to use a modified version of PostMe().
  If such a program wishes to reactivate before its gadget has been clicked
  in the TWM window, it should first call UnPostMe() (no arguments, no
  return) to inform TWM that the gadget should be taken down. Programs that
  will NOT need to call UnPostMe() (i.e. most programs) can use a version
  of twmClient.c from which the UnPostMe() function and all references to
  the global variable Delmsg have been removed.
    Before the first call to PostMe(), the function twmInit() must be
  invoked to set up the required messages and ports that PostMe() will need.
    Before the client exits, it should call the function UnPostMe() to
  deallocate resources twmInit() has allocated.
*/

#include "header/twm.h"
/* This invisible gadget lives in TWM's tiny window. When clicked on, the
   tiny window is removed and TWM's working window is opened.        */
struct Gadget WakeUpGadget = {
   NULL,              /* address of next gadget   */
   2, 10, 116, 10,    /* left, top, width, height  */
   GADGHNONE,         /* flags – no highlighting   */
   RELVERIFY,         /* activation flags          */
   BOOLGADGET,        /* gadget type               */
   NULL,              /* no imagery                */
   NULL,              /* no alternate imagery      */
   NULL,              /* no text                   */
   0,                 /* mutual exclude            */
   NULL,              /* SpecialInfo               */
   0,                 /* gadget ID                 */
   NULL,              /* user data                 */
};
struct NewWindow wtiny = {
   480, 60, 120, 20,  /* left, top, width, height    */
     0, 1,            /* detail pen, block pen       */
   GADGETUP           /* IDCMP flags                 */
   | CLOSEWINDOW,
   WINDOWDRAG         /* Window flags                */
   | WINDOWCLOSE
   | WINDOWDEPTH,
   &WakeUpGadget,     /* application gadget list     */
   NULL,              /* special checkmark imagery   */
   (UBYTE *)"TWM",    /* window title                */
   NULL,              /* custom screen pointer       */
   NULL,              /* super bitmap pointer        */
   0, 0, 0, 0,        /* min/max width and height    */
   WBENCHSCREEN       /* screen type                 */
};
struct NewWindow whuge = {
   480, 60, 0, 0,     /* left, top, width, height    */
     0, 1,            /* detail pen, block pen       */
   GADGETUP
   | CLOSEWINDOW,     /* IDCMP flags                 */
   WINDOWDRAG         /* Window flags                */
   | WINDOWCLOSE
   | WINDOWDEPTH
   | SMART_REFRESH,
   NULL,              /* application gadget list     */
   NULL,              /* special checkmark imagery   */
   (UBYTE *)"TWM",    /* window title                */
   NULL,              /* custom screen pointer       */
   NULL,              /* super bitmap pointer        */
   0, 0, 0, 0,        /* min/max width and height    */
   WBENCHSCREEN       /* screen type                 */
};
struct TextAttr twmFont = {   /* 80 column topaz font      */
   (UBYTE *)"topaz.font",
   TOPAZ_EIGHTY,
   FS_NORMAL,
   FPF_ROMFONT
};
SHORT borderlines[5][2] = { /* simple box around gadgets  */
   {-3,               -3             },
   {GADGWIDTH + 3,    -3             },
   {GADGWIDTH + 3,    GADGHEIGHT + 2},
   {-3,               GADGHEIGHT + 2},
   {-3,               -3             }
};
/* The following is the default contents of a client's gadget */
struct twmGadget gadgTemplate = {
   /* intuition gadget structure */
   NULL,                              /* address of next gadget      */
   0, 0, GADGWIDTH, GADGHEIGHT,       /* left, top, width, height    */
   GADGHCOMP,                         /* flags – invert to highlight */
   RELVERIFY,                         /* activation flags            */
   BOOLGADGET,                        /* gadget type                 */
   NULL,                              /* address of border struct    */
   NULL,                              /* SelectRender                */
   NULL,                              /* address of intuitext struct */
   0,                                 /* mutual exclude              */
   NULL,                              /* SpecialInfo                 */
   0,                                 /* gadget ID                   */
   NULL,                              /* user data                   */
   /* intuition border structure */
   0, 0,                              /* left edge, top edge         */
   2, 0, JAM1,                        /* front, back pens, draw mode */
   5,                                 /* number of points in border  */
   (SHORT *)borderlines,              /* address of coordinate array */
   NULL,                              /* address of next border      */
   /* intuitext structure */
   1, 0, JAM1,                        /* front, back pens, draw mode */
   0, 1,                              /* left edge, top edge         */
   &twmFont,                          /* address of TextAttr struct  */
   NULL,                              /* pointer to text             */
   NULL,                              /* address of next IntuiText   */
   /* name of gadget as supplied by client */
   "",
   /* pointer to message that requested this gadget */
   NULL
};

extern VOID *OpenWindow(), *OpenLibrary();
extern VOID *CreatePort(),  *FindPort();
extern VOID *GetMsg(),      *AllocMem();

struct IntuitionBase *IntuitionBase;
struct twmGadget *NewGadget();
struct MsgPort       *mp;   /* public port (called PORTNAME) */
struct NewWindow     *nw;   /* describes current window      */
struct Window        *w;    /* pointer to current window     */
struct twmMessage    *Tmsg; /* message arrived at mp         */
struct IntuiMessage  *Imsg; /* message arrived at IDCMP      */
struct twmGadget     *twmg; /* first gadget in my list       */

main ()
{
register int exitflag;                 /* quit input loop if set         */
register int swapflag;                 /* use other window (tiny/huge)   */
register int tinyflag;                 /* currently using tiny window    */
register UWORD class;                  /* IDCMP message class            */
UWORD code;                            /* IDCMP message code             */
int gadgCount;                         /* # of gadgets in my list        */
register struct twmGadget *gadget;     /* gadget clicked in huge window */
   exitflag   = FALSE;
   swapflag   = FALSE;
   tinyflag   = TRUE;                  /* start out with tiny window     */
   gadgCount  = 0;
   if ((IntuitionBase = OpenLibrary("intuition.library", 33L)) == NULL)
     CloseStuff(E_OPEN_INTUI);
   if (FindPort(PORTNAME) != NULL)  /* if we already exist, quit   */
     CloseStuff(E_ALREADY_UP);
   if ((mp = CreatePort(PORTNAME, 0L)) == NULL)
     CloseStuff(E_OPEN_PORT);
   if ((w = OpenWindow(nw = &wtiny)) == NULL)
     CloseStuff(E_OPEN_WINDOW);
   /* exitflag set by close gadget on tiny window if no current clients */
   while (!exitflag) {
     /* waiting for message at IDCMP or our own port */
     Wait(1L << w->UserPort->mp_SigBit | 1L << mp->mp_SigBit);

     while (Imsg = GetMsg(w->UserPort)) { /* check IDCMP messages first */
       class   = Imsg->Class;
       code    = Imsg->Code;
       gadget  = (struct twmGadget *)Imsg->IAddress;
       ReplyMsg(Imsg);
       if (class == CLOSEWINDOW)
         /* exit from tiny window only if we have no clients, else beep */
         if (tinyflag)
           if (gadgCount == 0)
             exitflag = TRUE;
           else
             DisplayBeep(w->WScreen);
         /* close gadget on huge window means switch back to tiny */
         else
           swapflag = TRUE;
       /* this message means gadget pressed in huge window  */
       else if (class == GADGETUP)
         if (tinyflag)
           swapflag = TRUE;
         else {
           gadget->tgMessage->tmAction = E_OK;       /* return code E_OK */
           KillGadget(gadget->tgMessage, TRUE);      /* get rid of gadget  */
           gadgCount--;
           swapflag = TRUE;                          /* switch to tiny     */
           ReplyMsg(gadget->tgMessage);              /* inform client      */
         }
     }
     /* now check messages at our public port */
     while (Tmsg = GetMsg(mp)) {
       /* client going on vacation, create a gadget for him  */
       if (Tmsg->tmAction == TWM_ACTION_ADD) {
         if ((gadget = NewGadget(Tmsg)) == NULL) {
           Tmsg->tmAction = E_NO_MEM;     /* send regrets */
           ReplyMsg(Tmsg);
         }
         else
           gadgCount++;
         /* if the huge window is up right now, close and re-open
            sothat we can be sure the new gadget will fit */
         if (!tinyflag) {
           SavePosCloseW(nw, w);
           CalcGadgPos(nw);
           if ((w = OpenWindow(nw)) == NULL)
```

```
                  CloseStuff(E_ABANDON_SHIP);
            }
         }
         /* client going right out of business, cancel his gadget */
         else if (Tmsg->tmAction == TWM_ACTION_DELETE) {
            /* kill the gadget, and ghost it if huge window is up */
            if (KillGadget(Tmsg, !tinyflag)) {
               Tmsg->tmAction = E_OK;
               gadgCount--;
            }
            else
               Tmsg->tmAction = E_TASK_UNKNOWN; /* unrecognized client */
            ReplyMsg(Tmsg);
         }
         /* some message type we don't know */
         else {
            Tmsg->tmAction = E_ACTION_UNKNOWN;
            ReplyMsg(Tmsg);
         }
      }
      if (swapflag) {      /* switch between huge and tiny windows */
         swapflag = FALSE;
         SavePosCloseW(nw, w);
         nw = tinyflag ? &whuge : &wtiny;
         tinyflag = !tinyflag;
         /* if we're going to open huge window, reformat
            gadgets and recalculate the window size   */
         if (!tinyflag)
            CalcGadgPos(nw);
         if ((w = OpenWindow(nw)) == NULL)
            CloseStuff(E_ABANDON_SHIP);
      }
   }
   CloseStuff(E_OK);
}

/* CloseStuff

   Close and deallocate everything. If there are any active clients, that
   means something has gone wrong, so we send them an E_ABANDON_SHIP.
   The return error codes start at 500 as defined in twm/header.h
*/
CloseStuff (error) int error;
{
register struct twmGadget *g;
   g = twmg;
   if (w)             CloseWindow(w);
   if (mp)            DeletePort(mp);
   if (IntuitionBase) CloseLibrary(IntuitionBase);
   while (g != NULL) {
      g->tgMessage->tmAction = E_ABANDON_SHIP;
      ReplyMsg(g->tgMessage);
      KillGadget(g->tgMessage);
   }
   exit(error);
}

/* NewGadget

   We have a new client to create a gadget for. We link him to the
   NextGadget field of the last gadget on the list, set up the new gadget
   and return its address.
*/
struct twmGadget *NewGadget (msg)
struct twmMessage *msg;
{
register struct twmGadget *g, *gprev;
register char *clientname;
register char c;
   gprev= NULL;
   g      = twmg;
   while (g != NULL) {
      gprev = g;
      g = g->tgMynext;
   }
   if ((g = AllocMem((long)sizeof(struct twmGadget), 0L)) == NULL)
      return FALSE;
   if (gprev != NULL) {
      gprev->tgMynext = g;
      gprev->tgGadget.NextGadget = &g->tgGadget;
   }
   *g = gadgTemplate;
   clientname = msg->tmName + strlen(msg->tmName);
   while (clientname > msg->tmName
         && (c = *(clientname - 1)) != ':'
         && c != '/')
      clientname--;
```

```
   strncpy(g->tgName, clientname, GADGNAMESIZE - 1);
   g->tgGadget.GadgetRender  = (APTR)&g->tgBorder;
   g->tgGadget.GadgetText    = &g->tgIText;
   g->tgIText.LeftEdge       = (GADGNAMESIZE - strlen(g->tgName)) << 2;
   g->tgIText.IText          = (UBYTE *)g->tgName;
   g->tgMessage              = msg;
   if (twmg == NULL)
      twmg = g;
   return g;
}

/* KillGadget

   Get rid of a gadget currently on our list. If off_flag is true, the
   gadget is currently being displayed, so we'll ghost it. Return FALSE
   if the gadget is not on the list.
*/
KillGadget (msg, off_flag)
struct twmMessage *msg; int off_flag;
{
register struct twmGadget *g, *gprev;
int flag;
   flag    = FALSE;
   gprev= NULL;
   g       = twmg;
   while (g != NULL && !flag)
      if (g->tgMessage->tmMessage.mn_ReplyPort ==
            msg->tmMessage.mn_ReplyPort)
         flag = TRUE;
      else {
         gprev= g;
         g      = g->tgMynext;
      }
   if (flag) {
      if (off_flag)
         OffGadget(&g->tgGadget, w, 0L);
      RemoveGadget(w, &g->tgGadget);
      if (gprev != NULL)
         gprev->tgMynext = g->tgMynext;
      else
         twmg = g->tgMynext;
      FreeMem(g, (long)sizeof(struct twmGadget));
   }
   return flag;
}

/* CalcGadgPos

   Position the gadgets in the huge window, and set the window
   size to accommodate them. The gadgets are displayed four
   across,  to the maximum depth of the screen.
*/
CalcGadgPos (nw)
register struct NewWindow *nw;
{
register int i, x, y;
register struct twmGadget *g;
   i  = 0;                      /* gadget counter        */
   x = GADGHGUTTER +   2;       /* starting x position   */
   y = GADGVGUTTER + 10;        /* starting y position   */
   g = twmg;                    /* address of 1st gadget */
   /* chain through gadget list, writing in new left and top   */
   while (g) {
      g->tgGadget.LeftEdge  = x;
      g->tgGadget.TopEdge   = y;
      /* if this gadget is in R.H. column, reposition to left of next line */
      if ((i++ & 3) == 3) {
         x = GADGHGUTTER + 2;
         y += GADGHEIGHT + GADGVGUTTER;
      }
      else
         x += GADGWIDTH + GADGHGUTTER;
      /* chain to next gadget */
      g = g->tgMynext;
   }
   /* if there are no gadgets, make the window big enough to hold 1 */
   if (i == 0)
      x = GADGHGUTTER * 2 + GADGWIDTH + 2;
   /* if less than 4 gadgets, make window just big enough to hold them */
   if (i < 4)
      nw->Width = x;
   else /* otherwise make it full width */
      nw->Width = GADGHGUTTER + 2 + (GADGHGUTTER + GADGWIDTH) * 4;
   /* if the last gadget on the list falls at the R.H. edge of
      the window, y is already big enough; otherwise, make it so */
   if ((i & 3) == 0)
      nw->Height = y;
```

```
    else
      nw->Height = y + GADGVGUTTER + GADGHEIGHT;
    /* make sure that new dimensions of window will still fit
      on the screen. . . if necessary reposition the window */
    if (nw->LeftEdge + nw->Width > w->WScreen->Width)
      nw->LeftEdge = w->WScreen->Width - nw->Width;
    if (nw->TopEdge + nw->Height > w->WScreen->Height)
      nw->TopEdge = w->WScreen->Height - nw->Height;
    /* install our first gadget as the new window's first gadget */
    nw->FirstGadget = &twmg->tgGadget;
}

/* SavePosCloseW

    Save the current window position and size in the NewWindow
    structure for that window, then close the window.
*/

SavePosCloseW (nw, w)
register struct NewWindow *nw;
register struct Window *w;
{
    nw->LeftEdge  = w->LeftEdge;
    nw->TopEdge   = w->TopEdge;
    nw->Width     = w->Width;
    nw->Height    = w->Height;
    CloseWindow(w);
}

/* When compiling with Aztec, the following two stubs replace the Aztec
    code for parsing the command line, thus reducing code size a bit */

#ifdef AZTEC_C
_wb_parse () {}
_cli_parse () {}
#endif !AZTEC_C
```

LISTING 2: HEADER/TWM.H

```
/* header/twm.h

    This is the same header file for both twm.c and twmClient.c.
    However, the intuition include is not needed for twmClient,
    and can be omitted to reduce compilation time.
*/

#include <intuition/intuitionbase.h>
#include <exec/types.h>
#include <exec/memory.h>
#include <exec/ports.h>
#include <exec/lists.h>

#define PORTNAME ("TinyWindowManager")
#define GADGNAMESIZE  17
#define GADGHGUTTER   18
#define GADGVGUTTER   10
#define GADGWIDTH     (GADGNAMESIZE << 3)
#define GADGHEIGHT    10
/* commands passed in twm_action field of a twmMessage */
#define TWM_ACTION_ADD      0
#define TWM_ACTION_DELETE  1
/* return codes passed back in same field */
#define E_OK                  0
#define E_OPEN_INTUI        501
#define E_ALREADY_UP        502
#define E_OPEN_PORT         503
#define E_OPEN_WINDOW       504
#define E_ACTION_UNKNOWN    505
#define E_TASK_UNKNOWN      506
#define E_NO_MEM            507
#define E_ABANDON_SHIP      508
struct twmMessage {
    struct Message tmMessage; /* Exec message structure */
    char *tmName;             /* the client's gadget name */
    int tmAction;             /* add or delete gadget     */
};
struct twmGadget {
    struct Gadget        tgGadget;                 /* the gadget for a client */
    struct Border        tgBorder;                 /* box around gadget       */
    struct IntuiText     tgIText;                  /* text in gadget          */
    char                 tgName[GADGNAMESIZE];     /* string for Intuitext    */
    struct twmMessage *tgMessage;                  /* msg to reply on click   */
    struct twmGadget   *tgMynext;                  /* my link to next gadget  */
};
```

LISTING 3: TEST1.C

```
/* test1.c

    This is a test program for TWM, and provides an example of using twmClient
    in an application. It puts up a window named "Press any key"; if a key is
    pressed, the window is taken down, and is replaced by either a tiny window
    or by a gadget in TWM's window (if PostMe() returns TRUE). Clicking in
    the main part of the tiny window kills it, and brings the big window back.
    The same applies to the gadget in the TWM window, if it is being used
    instead: clicking on it kills it and bring up this program's big window.
    Clicking close in either the big or the small window exits the program.
      The compiled test1 program may be copied to test2, test3 etc, and all
    copies run simultaneously, to get a better idea of how TWM works.
      Under Aztec, put this program and twmClient.c in the current directory,
    and twm.h in the subdirectory "header", then compile and link thus:

      cc +Htwm.p header/twm.h
      cc +Itwm.p test1
      cc +Itwm.p twmClient
      ln test1.o twmClient.o -lc
*/

#include "header/twm.h"
#define HUGE_WINDOW_NAME   ((UBYTE *)"Press any key")

struct NewWindow wtiny = {
    280, 120, 120, 20,      /* left, top, width, height       */
       0,    1,             /* detail pen, block pen          */
    MOUSEBUTTONS            /* IDCMP flags                    */
    | CLOSEWINDOW,
    WINDOWDRAG              /* Window flags                   */
    | WINDOWCLOSE
    | WINDOWDEPTH
    | SMART_REFRESH
    | ACTIVATE,
    NULL,                   /* application gadget list        */
    NULL,                   /* special checkmark imagery      */
    NULL,                   /* window title                   */
    NULL,                   /* custom screen pointer          */
    NULL,                   /* super bitmap pointer           */
    0, 0, 0, 0,             /* min/max width and height       */
    WBENCHSCREEN            /* screen type                    */
};
struct NewWindow whuge = {
    280, 120, 360, 60,      /* left, top, width, height       */
       0,    1,             /* detail pen, block pen          */
    CLOSEWINDOW             /* IDCMP flags                    */
    | RAWKEY,
    WINDOWDRAG              /* Window flags                   */
    | WINDOWCLOSE
    | WINDOWDEPTH
    | SMART_REFRESH
    | ACTIVATE,
    NULL,                   /* application gadget list        */
    NULL,                   /* special checkmark imagery      */
    HUGE_WINDOW_NAME,       /* window title                   */
    NULL,                   /* custom screen pointer          */
    NULL,                   /* super bitmap pointer           */
    0, 0, 0, 0,             /* min/max width and height       */
    WBENCHSCREEN            /* screen type                    */
};

extern VOID *OpenWindow(), *OpenLibrary(), *AllocMem(), *GetMsg();
struct IntuitionBase *IntuitionBase = NULL;
struct NewWindow  *nw    = NULL;
struct Window     *w     = NULL;
struct IntuiMessage *lmsg = NULL;

main (argc, argv)
int argc; char **argv;
{
int exitflag;    /* true when close gadget has been pressed          */
int swapflag;    /* true when it's time to switch between windows   */
int tinyflag;    /* true when the tiny window is up                  */
UWORD class; /* IDCMP message class                                  */
UWORD code; /* IDCMP message code                                    */
char filename[31];
register int i, c;
  /* Use program name as title of tiny window */
  for (i = strlen(argv[0]) - 1;
       i > 0 && (c = argv[0][i - 1]) != ':' && c != '/'; i--)
    ;
  wtiny.Title = (UBYTE *)(&argv[0][i]);
  if ((IntuitionBase = OpenLibrary("intuition.library", 33L)) == NULL)
    CloseStuff(E_OPEN_INTUI);
  /* Let twmClient do its allocations. . . we don't care this
```

```
    time, but it returns FALSE if the allocations fail   */
  twmInit();
  /* initialize flags */
  tinyflag = swapflag = exitflag = FALSE;
  /* open the big window, and save pointer to its NewWindow struct */
  if ((w = OpenWindow(nw = &whuge)) == NULL)
    CloseStuff(E_OPEN_WINDOW);
  while (!exitflag) { /* IDCMP loop */
    Wait(1L << w->UserPort->mp_SigBit);
    while (Imsg = GetMsg(w->UserPort)) {
      class= Imsg->Class;
      code= Imsg->Code;
      ReplyMsg(Imsg);
      if (class == CLOSEWINDOW)
        exitflag = TRUE;
      /* swap if tiny window clicked, or key pressed in big window */
      else if ((class == MOUSEBUTTONS && code == SELECTUP && tinyflag)
            || class == RAWKEY)
        swapflag = TRUE;
    }
    if (swapflag) {
      swapflag = FALSE;
      /* remember where this window is now stationed, and close it */
      SavePosCloseW(nw, w);
      /* if tiny window is now up, or PostMe() fails, open other window */
      if (tinyflag || !PostMe(wtiny.Title)) {
        nw = tinyflag ? &whuge : &wtiny;
        tinyflag = !tinyflag;
      }
      if ((w = OpenWindow(nw)) == NULL)
        CloseStuff(E_OPEN_WINDOW);
    }
  }
  CloseStuff(E_OK);
}

/* CloseStuff

  Call twmCleanUp() to deallocate messages and port we've been using,
  then close our own stuff and exit.
*/
CloseStuff (error)
int error;    /* errors start at 500 (see twm.h) except for E_OK = 0   */
{
  twmCleanUp();  /* twmClient deallocations */
  if (w)            CloseWindow(w);
  if (IntuitionBase)  CloseLibrary(IntuitionBase);
  exit(error);
}

/* SavePosCloseW

  Save the current window position and size in the NewWindow structure for
  that window, then close the window.
*/
static SavePosCloseW (nw, w)
register struct NewWindow *nw;
register struct Window *w;
{
  nw->LeftEdge= w->LeftEdge;
  nw->TopEdge= w->TopEdge;
  nw->Width    = w->Width;
  nw->Height   = w->Height;
  CloseWindow(w);
}
```

LISTING 4: twmClient.c

```
/* twmClient.c

  Tiny-Window Manager v1.0      (c) 1987 Transactor Publishing Inc.
  Client Interface Module
  by Nick Sullivan

  This program is freely redistributable provided that no charge is made
  for the redistribution beyond reasonable reproduction costs, and that no
  changes are made, except with the prior written approval of Transactor
  Publishing Inc., 85 West Wilmot St. #10, Richmond Hill, Ontario L4B 1K7.
  Programs that incorporate this code should include in their documentation
  the line: "This program supports TWM ((c) 1987 Transactor Publishing Inc.)"
    This module should be compiled and linked with applications that wish
  to be clients of TWM when it is present in the system. Briefly, the client
  calls the function twmInit() to set up, afterwards calls PostMe() whenever
  he wishes to go to sleep, then finally calls twmCleanUp() just before
  exiting. Full details are in the prefatory comments to TWM.c. Note that
  the size of this module can be further reduced, especially in programs
  that do not require UnPostMe(); for details see comments in the code.
*/

#include "header/twm.h"
#define TWM_MSGSIZE ((long)sizeof(struct twmMessage))
extern VOID *CreatePort(), *FindPort();
extern VOID *GetMsg(),    *AllocMem();
struct MsgPort          *mp      = NULL;  /* reply port for our msgs          */
struct MsgPort          *twmport = NULL;  /* points to twm's port             */
struct twmMessage       *Addmsg = NULL;  /* TWM_ACTION_ADD message           */
/* The following line can be deleted if UnPostMe() is not required.  */
struct twmMessage   *Delmsg  = NULL;  /* TWM_ACTION_DELETE message  */
int twmReady = FALSE;   /* TRUE means ports are allocated & initialized */

PostMe (clientName)
register char *clientName;
{
  /* trying not to pass junk to TWM. . . this if-statement can
     be deleted after the client application has been debugged. */
  if (clientName == NULL || *clientName == '\0')
     return FALSE;
  /* check we're initialized and that TWM exists in system. . .
     the first condition in the if-statement can be removed
     after the client application has been debugged       */
  if (!twmReady || (twmport = FindPort(PORTNAME)) == NULL)
     return FALSE;
  /* set up our message telling TWM to add its gadget   */
  Addmsg->tmName = clientName;
  Addmsg->tmAction = TWM_ACTION_ADD;
  PutMsg(twmport, Addmsg);
  /* The remainder of this function would need to be modified if
     the client wants to be able to re-awaken on some stimulus
     other than the user clicking its gadget in TWM's window. */
  WaitPort(mp);
  Addmsg = GetMsg(mp);
  /* anything other than E_OK return code is bad news. . . forget about TWM */
  return (Addmsg->tmAction == E_OK);
}

/* The following function can be deleted in normal use */
UnPostMe ()
{
  if (twmReady && (twmport = FindPort(PORTNAME)) != NULL) {
    Delmsg->tmAction = TWM_ACTION_DELETE;
    PutMsg(twmport, Delmsg);
    /* TWM will reply the original (ADD) message before replying this one
       if it's going to reply it at all. . . hence loop exit condition */
    do {
       WaitPort(mp);
    } while (GetMsg(mp) != Delmsg);
  }
}

/* This function must be called by the client before calling PostMe() */
twmInit ()
{
  if (twmReady)  /* don't re-initialize */
    return TRUE;
  /* set up our messages, allocate a port. The allocation of Delmsg
    can be deleted if the UnPostMe() function is not required   */
  if ((mp = CreatePort(NULL, 0L)) == NULL
     || (Delmsg = AllocMem(TWM_MSGSIZE, MEMF_CLEAR)) == NULL
     || (Addmsg= AllocMem(TWM_MSGSIZE, MEMF_CLEAR)) == NULL
    ) {
    twmCleanUp();
    return FALSE;
  }
  else {
    /* the next two lines can be deleted if UnPostMe() is not required. */
    Delmsg->tmMessage.mn_ReplyPort = mp;
    Delmsg->tmMessage.mn_Node.ln_Type = NT_MESSAGE;
    Addmsg->tmMessage.mn_ReplyPort = mp;
    Addmsg->tmMessage.mn_Node.ln_Type = NT_MESSAGE;
    return (twmReady = TRUE);
  }
}

/* This function must be called by the client before it exits. */
twmCleanUp ()
{
  twmReady = FALSE;
  if (mp)       DeletePort(mp);
  if (Addmsg)  FreeMem(Addmsg, TWM_MSGSIZE);
  /* the next line can be deleted if UnPostMe() is not required. */
  if (Delmsg)   FreeMem(Delmsg, TWM_MSGSIZE);
}
```

The View Port

By Larry Phillips

Larry Phillips is an Amiga hardware and software hacker from Vancouver, British Columbia, and a SYSOP on CompuServe's AmigaForum. This is the first instalment of a regular column in which Larry talks about. . . well, whatever is on his mind. You may not always agree with his opinions, but we hope you'll enjoy his discussions on a wide variety of mostly Amiga-related topics.

Do you remember when the personal computer came onto the scene? When the first home colour machines became available? The first hard disk drives? How about software? Desktop publishing? All of these advancements in the capabilities of the home computer have been welcomed by enthusiasts. For months or years, the buzzwords flew, refinements were anticipated, people wondered when their own machine would be capable of such marvels.

There is a new wave of buzzwords now, and you will be hearing them more and more in the next few years. The buzzwords have to do with "multitasking". The reason you will hear more about this is twofold. The first is that machines are becoming powerful enough, and fast enough, to make it worthwhile. The second reason is that the big players in the game will be introducing multitasking machines, and of course will, as usual, claim not only that they invented the technique, but that their particular implementation is the best. We all know who these companies are, and in fact they have already started the wheels in motion. IBM has announced the PS/2 line, and Apple has announced their Mac II. Of course the operating system for the PS/2 line is still vapour, and the Mac doesn't have the MMU yet, but they are coming.

I am confident that IBM will pretend they invented multitasking, just as they did with "Virtual Storage" on their mainframes. They may not come right out and say so, but the people who buy computers from them will look around after the fact and say, "Look at all those other computer companies imitating IBM. Aren't we great for buying our machines from the company that paved the way?"

Apple has already gone even farther. A sign on their booth at a trade show recently proclaimed the Mac II to be "The first in a new generation of multitasking home computers". Ask any Mac II owner and he will tell you that the Mac II has multitasking capability. What he won't tell you is his definition of the buzzwords. With the current software on the Mac II, one can indeed multitask, but only if the program that is currently running specifically relinquishes control to the supervisory scheduling program, or if the user manually causes a context switch to another, waiting application.

Let's take a look at a machine that really does multitask, one that has been doing so for two years now. Of course I am referring to the Amiga. The Amiga is not the first multitasking home computer, but it does have one distinction from previous multitasking systems such as the CoCo with the OS/9 operating system. That distinction is all important, and is the fact that programs need not be specially written to use multitasking. This means that a program, unless written to inhibit multitasking, will get along fine with other programs running concurrently.

I have heard multitasking referred to as a "gimmick", as "useless", and worse. I have heard statements to the effect that "I can only do one thing at a time anyway", or "I would never use it". One of the nicest things I can say about multitasking is that I hardly ever notice it any more. I don't think of it as a "feature" – multitasking has become an ordinary, everyday activity for me, and I only appreciate it properly when I am faced with the unpleasant prospect of using a single–tasking machine.

With multitasking, you need not sit and wait while a compile or download is happening, or worry about whether you loaded up that favourite "resident utility". Your favourite resident utilities are there. All the time. All of them. Even your not–so–favourite ones are there, ready to be called upon. Need to look up a name and address? Uh oh. . . the compiler is running! No problem, just push the compiler window into the background and do a directory, load up an editor, a phone book, and address book, or in fact anything you want. The only limitation is the amount of memory you have.

But won't everything run slower?

I'm very glad you asked that question, because it is one that can be answered with an example. You don't get anything for nothing. There's no such thing as a free lunch. We all know that, so where's the catch? The answer is that there is no catch. You do get something for nothing. What you get is better utilization of the available machine cycles. You get to use the resources you paid for when you bought your machine. Consider the following reasonably typical scenario. We will look at it on two different machines, one being the Amiga, and another being a mythical single–tasking machine that just happens to run the same programs and at the same speed.

First the single–tasker. We will consider three programs: a terminal program, an editor, and a "CPU bound" program, say something like a Mandelbrot picture generator. With the terminal program we are going to download a file that will take about an hour at 300 baud. With the editor we are going to write the all time best novel/computer game/whatever, and will spend an hour editing. The Mandelbrot generator will do the picture we want in an hour (you *are* papering your computer room walls with 'brots aren't you?). The fact that the times are all one hour is an incredible coincidence, but will serve our purposes.

On the single–tasking machine, we load up our terminal program, call the network or BBS, start the download, and go have lunch. When it is done, we can call up the editor and spend an hour writing. Finally, we start the Mandelbrot generator and wait again. Total time for these activities is three hours, not counting any time we spent setting up each program. Three hours, in which we spent exactly one hour interacting with the computer, and two hours waiting for it to finish.

On the Amiga, things are far better. First, we start the Mandelbrot generator, setting it to a priority of –2. Don't worry too much if "–2" doesn't mean much to you yet. It's a magic number that tells the Amiga something about your wishes.

A little more on it later. Now, the good part starts. Right away, the first reaction of a new Amiga owner is to sit and wait for it to finish. Not so for the veteran! The veteran immediately starts up the terminal program, setting it to a priority of 0 (another magic number, the Amiga is full of them). The BBS or network is called, and the download started. Pushing the terminal program back out of the way, we finally call the editor, at a priority of –1, and start on that program or manuscript.

By now you've probably guessed that the Amiga is going to be busy for less than the three hours required by the other machine, and of course you're right. What happens is this: the three magic numbers tell the Amiga that the terminal program (priority 0) has priority over the other two programs, and that the editor has priority over the Mandelbrot program. The Mandelbrot program can say anything it wants to the potted plant on the desk.

The result of this priority scheme is that whenever the terminal program needs to get and process characters from the keyboard or serial port, it will be allowed to do so, usually immediately, and at worst, within a few milliseconds. Likewise the editor program will only be active when it has to service the keyboard or perform a command but, in addition, it will only be active when the terminal program is not actually processing data. Since characters are coming in slowly relative to the speed of the CPU, even at 1200 baud, the editor gets a lot of time to do its thing. Similarly, when the editor is not busy processing, and the terminal is not busy, the Mandelbrot program can churn out some more of its picture. The Mandelbrot program is too busy to allow the potted plant any time at all.

The bottom line is that the terminal program runs very nearly at full speed. It will be finished its download in about an hour. While it was happening, you were writing, and noticed no slowdown in the response of your editor, so for all intents and purposes, you got just as much done in the hour as you would have otherwise. Hmm. . . pretty good so far. We have done two hours' worth of computing in just one hour. But what about the poor Mandelbrot program? Did it get anything done? Of course it did! Neither downloading and editing are very CPU intensive, and the Mandelbrot program got a lot of cycles. Granted, it did not get an hour's worth of cycles, but it got a lot. In practical terms, the Mandelbrot program will be about half or three quarters of the way through, and will run for another fifteen minutes to half an hour. It will run longer if you are a fast typist, using more CPU cycles to service the keyboard input, and shorter if you type like I do.

So, in somewhat under 1.5 hours, we have done three hours' worth of work. I'd say that's a pretty good benefit, and when you consider that even while doing all this, we were not prevented from calling up some other job, or from looking at another file, listing disk contents, performing calculations etc., it becomes something I, at least, am not willing to do without on a home computer. I think it's called "being spoiled".

In case I sounded a bit upset back there when talking about IBM and Apple, let me say now that I am glad they are finally seeing the light. Their multitasking, when it arrives, will effectively "legitimize" the feature, and we Amiga owners can all sit back with very smug expressions indeed. Of course, being polite, we will not laugh uproariously when next year some IBM owner tells us how great it is to be able to run four programs at once. Will we?

ACCESS

The best of non-commercial software

by Steve Ahlstrom, Denver, Colorado

This new regular column is a natural for Steve Ahlstrom – he's the primary sysop on CompuServe's AmigaForum, which, by the way, gets a lot of traffic by avid Amiga users and programmers. However, AmigaForum won't be Steve's only source and you can bet he'll report any item worth knowing about. So if you have some information you'd like to share, you can leave a message to [76703,2006] on AmigaForum, or send it to us and we'll pass it along to Steve.

In the world of freely distributable software for the Amiga, there are many gems to be found. There is also a lot of broken glass. This column is meant to help you find the more useful and worthwhile of these programs.

First of all, I guess I should try to define just what "freely distributable" means. There are generally 3 classes of "freely distributable" software.

Public Domain

"Public Domain", on the surface, seems to be the easiest to describe. In reality, it's probably the hardest (at least as far as US copyright law goes).

To the non-legal type, "Public Domain" means that the author of the work (in our case, the author of an Amiga program) has given up all rights to the work and has placed his/her work into the "Public Domain" for all to use, enjoy, change, sell, or whatever the case may be. The intent is that the author claims no rights at all on the work. Under US copyright law, though, it's a little less clear-cut than that. The law does not specifically say that something can be placed into the "Public Domain" other than a work which has gone beyond the time frame granted for copyrights. There are many rules about how authors may reclaim their rights (within a specific time period) if for some reason they feel that they have lost control of those rights. If someone can place a program into the "Public Domain" then later re-establish control of the rights they explicitly gave up earlier, was it indeed "Public Domain" to begin with? See the possible problems?

When it is stated that a program is "Public Domain", the intent of the author *usually* is that the program may be freely distributed and the author has no intention of reclaiming rights or limiting distribution in any way.

Shareware

"Shareware" is a little more clear cut. It's merely a non-conventional method of marketing a program. In some cases, it works well, but in the vast majority of cases it is not very effectual.

With "Shareware", the author is retaining all of the rights to the work and is allowing distribution of the work via electronic networks, BBSs, user groups and so on. Somewhere in the documentation for the program you'll find a statement of the author's stipulations for distribution. Normally it says something to this effect: "If you like this program and find it useful, send me $xx.xx. If you do not, feel free to distribute this further, but please do not continue to use it if you have not paid for it." If you do send the author money, you will usually be put on a mailing list to inform you of newer versions of the program and how to get them. Sometimes you'll be sent, upon payment, the latest version and the printed documentation. Read the stipulations carefully because each program is a little different.

"Shareware" is a good concept, but like many good concepts, it has many abuses. It came about for a variety of reasons. The foremost reason is that someone thought of a need for a program but, because the program didn't have mass market appeal, no mainline software publisher would pick it up. By making it "Shareware" the author can get some money for the time and effort spent creating the program. "Shareware" usually fills a void between "Public Domain" and commercial software in quality. Unfortunately, many people are now putting the "Shareware" label on everything they write. The problem becomes that everyone is asking for money. Many people have decided that since they don't have to buy the program in a store there is no need to send in the money for it (even if they find the program useful and often used) because everyone seems to be asking for money. This is a decision only you can make. By not monetarily supporting those "Shareware" programs you find useful, you are discouraging the "Shareware" author from using this method of marketing for other works. Many "Shareware" programs really deserve support.

Other

The 3rd class is really just a subclass of "Shareware". It's a program that is copyrighted but for which the author asks no compensation. The author retains the rights to the program, often because he wants to leave his options open legally to stop someone else from selling his work for a profit when his original intent was for the program to be free.

SetPri: Adjust the priority of active tasks

SetPri by Ben Blish is a fine example of a needed program that has little commercial marketability. To quote Ben's documentation for SetPri:

> "SetPri is *NOT* Shareware; far be it from me to presume upon you. If you *really* like the program, just tell me so by mail or on compuserve. If you have suggestions, you can give those also. . . if you want to complain, call Commodore. I don't want to hear it! <grin>."

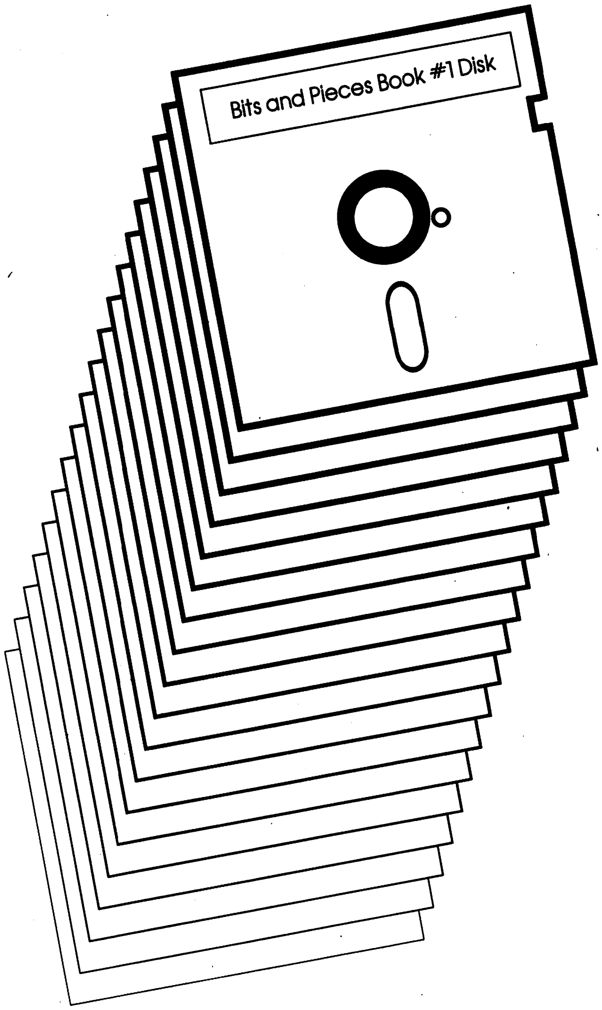
SetPri allows you to set or change the priority of any task currently active in the system. When SetPri is run, it brings up a small window showing all of your current tasks. If you have more than eight tasks active, you can scroll through the list with the proportional slider. Just click on the task name (program name) you wish to change and set the new task priority, either higher or lower.

There are many times (like now) that I may have my Amiga doing many things at once. Right now, I'm compiling a program, downloading a program from the AmigaForum on CompuServe, and typing this column. Since the compiler is doing frequent disk accesses, and the priority is (normally) set higher than either my term program or text editor, I was finding that I would have characters 'dropped' when typing. So, I ran SetPri (which has found a permanent home in my c: directory). I bumped up the priority of my term program from 0 to 5 and of my editor from 0 to 2.

With terminal programs and text editors, very little CPU time is used. Only when you are actually typing do the programs need to run. In CPU time, there is a huge amount of time for the system to use between keystrokes (even if you are 150 WPM typist!) Since I'm running the compiler in the background, it is running marginally slower than it would if it were the only task running but, because of resetting the tasks' priority levels, the compiler isn't interfering with either my term program or text editor. To me, controllable multitasking is the best thing the Amiga has going for it, and SetPri makes it easy to obtain that control.

Next Issue

There are dozens of freely distributable programs that can make life a lot easier for you. I'm sure you know about many of them, especially if you frequent the commercial networks and local BBSs. Some of them may have slipped past your notice. There are many freely distributable games, terminal programs, languages, text editors, spell checkers – almost anything you can imagine. I'll be telling you about many of them in future editions of this column. If there is something specific you'd like to see covered, just let me know!



News BRK

Submitting NEWS BRK Press Releases

If you have a press release you would like to submit for the NEWS BRK column, make sure that the computer or device for which the product is intended is prominently noted. We receive hundreds of press releases for each issue, and ones whose intended readership is not clear must unfortunately go straight to the trash bin. It should also be mentioned here that we only print product releases which are in some way applicable to Commodore equipment. News of events such as computer shows should be received at least 6 months in advance.

Transactor News

Transactor for the Amiga

In case you haven't heard, the premiere issue of *Transactor for the Amiga* will be released this January. The following few items describe some extra special offers that you shouldn't miss – most expire January 1st, 1988.

Half Price Until January 1st!

In case that looks like a typo, here it is repeated. Subscriptions to *Transactor for the Amiga* will be HALF PRICE until January 1st, 1988. Send just $7.50 US or $9.50 Cdn for a full year of *Transactor* for Amiga programmers!

Disk subscription prices are even more incredible! Until January 1st, 1988, a Transactor for the Amiga disk subscription will be just $29.95 US or $36.95 Cdn! That's nearly half off the regular price! You'll get a disk with every magazine containing all the programs we publish for the Amiga, and we'll probably add extra programs that aren't published too.

After January 1st, 1988, regular subscription prices for *Transactor for the Amiga* will be in effect. Magazine subscription prices are the same as for the original *Transactor* – $15 US and $19 Canadian. Disk subscriptions for the new mag will be a little more until the price of 3½ inch disks comes down – $55.00 US and $67.00 Cdn.

I Want To Switch!

With the *Transactor for the Amiga* coming this January, many of our readers will want to switch to the new mag for the higher concentration of Amiga material. Naturally there will be others that won't, and still others that will want both magazines.

To switch to the new magazine, there's no charge. Simply put your name and subscription number on our postage paid card and check off the appropriate box. Please don't omit your name as this gives us a cross-reference to ensure we change the correct subscriber record.

Remember, there's one more *Transactor* coming out before the first issue of *Transactor for the Amiga*. As usual, it will also contain Amiga material. But if this issue marks your last one, you'll not only need to switch, you'll need to renew as well.

I Want Both!

For those who want both magazines, we highly recommend you take advantage of the pre-release subscription pricing for the new Amiga publication. After January 1st, we'll be offering special combination subscription pricing, but it won't match the nice pre-release deals we're offering now. Details to be announced next issue.

Subscription Renewals and Enquiries

We get a lot of calls at the office. One common reason is to enquire about merchandise that has not been received. So before calling about a subscription order or other purchase, please try to follow these guidelines:

- First of all, make sure you contact the right people; if you receive *Transactor* with the special TPUG insert, you should call TPUG, not Transactor Publishing. If you renew a subscription with *Transactor*, you will receive the magazine without the insert, and your TPUG membership will not be renewed.
- Second, note the expiry date of your subscription; the Volume and Issue number of your last issue is shown on the first line of your mailing label.
- If you're a subscriber, have your subscription number ready – it's the fastest was for us to check.

Mail Order Products:

- Clearly print your full name, mailing address, phone number, and a CompuServe account number if you have one. Confirming an address or potential error is accomplished mush faster over the phone, voice or data.
- For renewals, include your subscription number AND at least one other piece of identifiable information such as your name. This allows us to ensure we extend the right persons subscription. For new subscriptions, please include your address. Believe it or not, we've actually received orders for subscriptions and other products with no address – not only are we unable to comply, but we can't even send back the uncancelled cheque!
- Ontario residents: remember to add 7% provincial sales tax. The reply card clearly shows which items are taxable.
- Make sure payment is enclosed or credit card number is included, or we will not be able to fill the order. If paying by credit card, clearly print ALL the numbers of the card, and include the expiry date.
- Do not staple or glue subscription cards; cards sealed in this way can get destroyed when opened, causing great problems in filling the order. It's best to use tape around the three open edges.

The 20/20 Deal

. . .is still in effect – order 20 subscriptions to the mag or disk, 20 back issues, 20 disks, etc., and get a 20% discount. However, this offer cannot be combined with other specials, or to TPUG subscriptions and products.

Transactor Special Offer

The special offer this issue is for Transactor Back Issues. Order any back issue at the regular price of $4.50 (US/C), and get additional back issues for only $2.00 each! Order 10 total and the effective price per copy is cut by half! ($4.50 + 9 × $2.00 = $22.50) Once again, this special offer is in effect for this issue only so it applies only to orders postmarked before January 1, 1988.

Transactor Mail Order

The following details are for products listed on the mail order card. If you have a particular question about an item that isn't answered here, please write or call. We'll get back to you and most likely incorporate the answer into future editions of these descriptions so that others might benefit from your enquiry.

■ Moving Pictures - the C-64 Animation System, $29.95 (US/C)
This package is a fast, smooth, full-screen animator for the Commodore 64, written by AHA! (Acme Heuristic Applications!). With Moving Pictures you use your favourite graphics tool to draw the frames of your movie, then show it at full animation speed with a single command. Movie 'scripts' written in BASIC can use the Moving Pictures command set to provide complete control of animated creations. BASIC is still available for editing scripts or executing programs even while a movie is being displayed. Animation sequences can easily be added to BASIC programs. Moving Pictures features include: split screen operation – part graphics, part text – even while a movie is running; repeat, stop at any frame, change position and colours, vary display speed, etc; hold several movies in memory and switch instantly from one movie to another; instant, on-line help available at the touch of a key; no copy protection used on disk.

■ The Potpourri Disk, $17.95 US, $19.95 Cdn.
This is a C-64 product from the software company called AHA!, otherwise known as Nick Sullivan and Chris Zamara. The Potpourri disk is a wide assortment of 18 programs ranging from games to educational programs to utilities. All programs can be accessed from a main menu or loaded separately. No copy protection is used on the disk, so you can copy the programs you want to your other disks for easy access. Built–in help is available from any program at any time with the touch of a key, so you never need to pick up a manual or exit a

program to learn how to use it. Many of the programs on the disk are of a high enough quality that they could be released on their own, but you get all 18 on the Potpourri disk for just $17.95 US / $19.95 Canadian. See the Ad in this issue for more information.

■ TransBASIC II $17.95 US, $19.95 Cdn.
An updated TransBASIC disk is now available, containing all TB modules ever printed. The first TransBASIC disk was released just as we published TransBASIC Column #9 so the modules from columns 10, 11 and 12 did not exist. The new manual contains everything in the original, plus all the docs for the extras. There are over 140 commands at your disposal. You pick the ones you want to use, and in any combination! It's so simple that a summary of instructions fits right on the disk label. The manual describes each of the commands, plus how to write your own commands.

People who ordered TB I can upgrade to TB II for the price of a regular Transactor Disk (8.95/9.95). If you are upgrading, you don't necessarily need to send us your old TB disk; if you ordered it from us, we will have your name on file and will send you TB II for the upgrade price. Please indicate on the order form that you have the original TB and want it upgraded.

Some TBs were sold at shows, etc, and they won't be recorded in our database, If that's the case, just send us anything you feel is proof enough (e.g. photocopy your receipt, your manual cover, or even the diskette), and TB II is yours for the upgrade price.

■ The Amiga Disk, $12.95 US, $14.95 Cdn.
Finally, the first Transactor Amiga disk is available. It contains all of the Amiga programs presented in the magazine, of course, including source code and documentation. You will find the popular "PopColours" program, the programmer's companion "Structure Browser", the Guru–killing "TrapSnapper", user-friendly "PopToFront", and others. In addition, we have included public domain programs – again, with documentation – that we think *Transactor* readers will find useful. Among these are the indispensable ARC; Csh, a powerful CLI–replacement DOS shell; BLink, a linker that is much faster and has more features than the standard ALink; Foxy and Lynx, a 6502 cross assembler and linker that makes its debut on the Amiga Disk; and an excellent shareware text editor called UEdit. In addition, we have included our own expression–evaluator calculator that uses variables and works in any number base. All programs contain source code and documentation; all can be run from the CLI, and some from Workbench. There's something for everyone on the Transactor Amiga disk.

■ Transactor T-Shirts, $13.95 US, $15.95 Cdn.
■ Jumbo T-Shirt, $17.95 US, $19.95 Cdn.
As mentioned earlier, they come in Small, Medium, Large, Extra Large, and Jumbo. The Jumbo makes a good night-shirt/beach-top – it's BIG. I'm 6 foot tall, and weigh in at a slim 150 pounds – the Small fits me tight, but that's how I like them. If you don't, we suggest you order them 1 size over what you usually buy.

One of the free gift choices we offer when you order a combination magazine AND disk subscription is a Transactor T-Shirt in the size and colour of your choice (sorry, Jumbo excluded). The shirts come in red or light blue with a 3-colour screen on the front featuring our mascot, Duke, in a snappy white tux and top hat, standing behind our logo in 3D letters.

■ Inner Space Anthology $14.95 US, $17.95 Cdn.
This is our ever popular Complete Commodore Inner Space Anthology. Even after two years, we still get inquiries about its contents. Briefly, The Anthology is a reference book – it has no "reading" material (ie. "paragraphs"). In 122 compact pages, there are memory maps for 5 CBM computers, 3 Disk Drives, and maps of COMAL; summaries of BASIC commands, Assembler and MLM commands, and Wordprocessor and Spreadsheet commands. Machine Language codes and modes are summarized, as well as entry points to ROM routines. There are sections on Music, Graphics, Network and BBS phone numbers, Computer Clubs, Hardware, unit-to-unit conversions, plus much more. . . about 2.5 million characters total!

■ The Transactor Book of Bits and Pieces #1, $14.95 US, $17.95 Cdn.
Not counting the Table of Contents, the Index, and title pages, it's 246 pages of Bits and Pieces from issues of *Transactor*, Volumes 4 through 6. Even if you have all those issues, it makes a handy reference – no more flipping through magazines for that one bit that you just know is somewhere. . . Also, each item is forward/reverse referenced. Occasionally the items in the Bits column appeared as updates to previous bits. Bits that were similar in nature are also cross-referenced. And the index makes it even easier to find those quick facts that eliminate a lot of wheel re-inventing.

■ The Bits and Pieces Disk, $8.95 US, 9.95 Cdn.
■ Bits Book AND Disk, $19.95 US, 24.95 Cdn.
This disk contains all of the programs from the Transactor book of Bits and Pieces (the "bits book"), which in turn come from the "Bits and Pieces" section of past issues of the magazine. The "bits disk" can save you a lot of typing, and in conjunction with the bits book and its comprehensive index can yield a quick solution to many a programming problem.

■ The G-LINK Interface, $59.95 US, 69.95 Cdn.
The Glink is a Commodore 64 to IEEE interface. It allows the 64 to use IEEE peripherals such as the 4040, 8050, 9090, 9060, 2031, and SFD-1001 disk drives, or any IEEE printer, modem, or even some Hewlett-Packard and Tektronics equipment like oscilloscopes and spectrum analyzers. The beauty of the Glink is its "transparency" to the C64 operating system. Some IEEE interfaces for the 64 add BASIC 4.0 commands and other things to the system that sometimes interfere with utilities you might like to install. The Glink adds nothing! In fact it's so transparent that a switch is used to toggle between serial and IEEE modes, not a linked-in command like some of the others. Switching from one bus to the other is also possible with a small software routine as described in the documentation.

As of Transactor Disk #19, a modified version of Jim Butterfield's "COPY–ALL" will be on every disk. It allows file copying from serial to IEEE drives, or vice versa.

■ The Tr@ns@ctor 1541 ROM Upgrades, $59.95 US, $69.95 Cdn.
You can burn your own using the ROM dump file on Transactor Disk #13, or you can get a set from us. There are 2 ROMs per set, and they fix not only the SAVE@ bug, but a number of other bugs too (as described in P.A. Slaymaker's article, Vol 7, Issue 02). Remember, if SAVE@ is about to fail on you, then Scratch and Save may just clobber you too. This hasn't been proven 100%, but these ROMs will eliminate any possibilities short of deliberately causing them (ie. allocating or opening direct access buffers before the Save).

NOTE: Our ROM upgrade kit does NOT fit in the 1541C drives. Where we supply two ROMs, Commodore now has it down to one MASSIVE 16 Kbyte ROM. We don't know if the new drives still contain the bugs eliminated by our kit, but we'll find out and re-cut a second kit if necessary. In the meantime, 1541C owners should not order this item until further notice.

■ The Micro Sleuth: C64/1541 Test Cartridge, $99.95 US, $129.95 Cdn.
We never expected this cartridge, designed by Brian Steele (a service technician for several schools in southern Ontario), would turn out to be so popular. The Micro Sleuth will test the RAM of a C64 even if the machine is too sick to run a program! The cartridge takes complete control of the machine. It tests all RAM in one mode, all ROM in another mode, and puts up a menu with the following choices:

1) Check drive speed
2) Check drive alignment
3) 1541 Serial test
4) C64 serial test
5) Joystick port 1 test
6) Joystick port 2 test
7) Cassette port test
8) User port test

A second board (included) plugs onto the User Port; it contains 8 LEDs that let you zero in on the faulty chip. Complete with manual.

Transactor Disks, Transactor Back Issues, and Microfiche

All *Transactors* since Volume 4 Issue 01 are now available on microfiche. According to Computrex, our fiche manufacturer, the strips are the "popular 98 page size", so they should be compatible with every fiche reader. Some issues are ONLY available on microfiche – these are marked "MF only". The other issues are available in both paper and fiche. Don't check both boxes for these unless you want both the paper version AND the microfiche slice for the same issue.

To keep things simple, the price of Transactor Microfiche is the same as magazines, both for single copies and subscriptions, with one exception: a complete set of 24 (Volumes 4, 5, 6, and 7) will cost just $49.95 US, $59.95 Cdn.

This list also shows the "themes" of each issue. Theme issues didn't start until Volume 5, Issue 01. Transactor Disk #1 contains all programs from Volume 4, and Disk #2 contains all programs from Volume 5, Issues 1-3. Afterwards there is a separate disk for each issue. Disk 8 from The Languages Issue contains COMAL 0.14, a soft-loaded, slightly scaled-down version of the COMAL 2.0 cartridge. And Volume 6, Issue 05 lists the directories for Transactor Disks 1 to 9.

■ Vol. 4, Issue 01 (■ Disk 1)
■ Vol. 4, Issue 02 (■ Disk 1)
■ Vol. 4, Issue 03 (■ Disk 1)
■ Vol. 4, Issue 04 – MF only (■ Disk 1)
■ Vol. 4, Issue 05 – MF only (■ Disk 1)
■ Vol. 4, Issue 06 – MF only (■ Disk 1)
■ Vol. 5, Issue 01 – Sound and Graphics (■ Disk 2)
■ Vol. 5, Issue 02 – Transition to Machine Language – MF only (■ Disk 2)
■ Vol. 5, Issue 03 – Piracy and Protection – MF only (■ Disk 2)
■ Vol. 5, Issue 04 – Business & Education – MF only (■ Disk 3)
■ Vol. 5, Issue 05 – Hardware & Peripherals (■ Disk 4)
■ Vol. 5, Issue 06 – Aids & Utilities (■ Disk 5)
■ Vol. 6, Issue 01 – More Aids & Utilities (■ Disk 6)
■ Vol. 6, Issue 02 – Networking & Communications (■ Disk 7)
■ Vol. 6, Issue 03 – The Languages (■ Disk 8)
■ Vol. 6, Issue 04 – Implementing The Sciences (■ Disk 9)
■ Vol. 6, Issue 05 – Hardware & Software Interfacing (■ Disk 10)
■ Vol. 6, Issue 06 – Real Life Applications (■ Disk 11)
■ Vol. 7, Issue 01 – ROM / Kernel Routines (■ Disk 12)
■ Vol. 7, Issue 02 – Games From The Inside Out (■ Disk 13)
■ Vol. 7, Issue 03 – Programming The Chips (■ Disk 14)
■ Vol. 7, Issue 04 – Gizmos and Gadgets (■ Disk 15)
■ Vol. 7, Issue 05 – Languages II (■ Disk 16)
■ Vol. 7, Issue 06 – Simulations and Modelling (■ Disk 17)
■ Vol. 8, Issue 01 – Mathematics (■ Disk 18)
■ Vol. 8, Issue 02 – Operating Systems (■ Disk 19)
■ Vol. 8, Issue 03 – Feature: Surge Protector (■ Disk 20)

Industry News

World Of Commodore Show

This year's World of Commodore Show promises to be as great as in previous years, and we'll be there as usual. Show organizers tell us that the original floor space allotment is 90% taken already and that more is being arranged.

Unlike previous years, the official show hotel is now the Airport Skyline. It's a bit farther from the International Center, but the Skyline runs a regular shuttle bus service to and from the show – a nice change!

Special show rates have been arranged – for more information contact Airport Skyline reservations at (416) 244-1711.

New England 1987 Commodore-Specific Computer Fair

MARCA New England is hosting this event on November 14 and 15, 1987 at Park West Hotel and Club, Marlboro, MA. The Fair will include vendor exhibits of Commodore 64, 128 and Amiga products; seminars by nationally-known speakers on beginner users, telecommunications, languages, graphics, music and more; public domain software and resource tables featuring hundreds of disks of PD material for the Amiga and experts on hand to answer questions. General admission is $15 for 2 days with unlimited seminars; one day admission is $8 without access to the seminars, $2 extra for seminar priviledges. MARCA members should contact their user group for special rates.

Contact: Frank Ordway, 6 Flagg Road, Marlboro, MA 01752, (617) 485-4677.

Also planned for the fair is "A Special Salute To Commodore" banquet on Friday, November 13. Al Duncan, President of Commodore, and Jim Butterfield, Commodore's most famous guru, will be the guests of honour. Cost for the banquet is $18.95. Advance registration is required; make cheque payable to SF Productions and send, along with name address and phone number, to: Sam Evangelous, 74 West Street, Clinton, MA 01510

The 64 Emulator for Amiga

ReadySoft Inc. announces "The 64 Emulator" for the Amiga. Simply insert the Emulator disk into your Amiga and your Amiga becomes a Commodore 64. The 64 Emulator has full support for all Amiga disk drives and printers, and with an optional interface cable, any Commodore 64 disk drive can be connected directly to your Amiga through the parallel printer port. All video modes, including sprites and raster interrupts (used in most games), as well as sound and colour are fully supported. A monochrome option is included for additional speed when colour is not required. Written fully in 68000 machine code for maximum speed, The 64 Emulator takes full advantage of the Amiga's hardware to give excellent 64 compatibility. Price is $39.95 US ($49.95 Cdn.), $59.95 US ($79.95 Cdn.) with interface cable. For further information contact: ReadySoft Inc., P.O. Box 1222, Lewiston, NY, 14092, (416) 731-1472.

Commodore Creating Product List

The following is from a letter from Peter Baczor, Commodore Customer relations manager, to manufacturers of products for Commodore machines:

> *In an effort to better support our end-users I am creating a specification library of products available for use with our computers, the 64-C, C-128, Amiga 500, Amiga 1000, Amiga 2000, and PC-10.*
>
> *I would be greatly appreciative if you would send me specification sheets for the products currently available that will operate with any of the abovementioned machines. Please send this information to the following address:*
>
> *Commodore Business Machines*
> *Customer Relations, Dept. SU87*
> *1200 Wilson Drive*
> *West Chester, PA 19380*
>
> *Thank you.*
>
> *Regards,*
> *Peter J Baczor*
> *Manager, Customer Relations*

New UEDIT Release

Uedit V2.3 is now available. Uedit is a Shareware text editor for the Amiga, noted for editing power (up to 100 files at once) and extreme versatility. It uses mouse cursor-placement and mouse-scrolling, function keys, menus, gadgets, and on-line help facility. All of these and virtually everything about Uedit can be customized while you are using it. Up to 1680 user-defined menu selections and hundreds of key, mouse, and gadget commands can be on-line.

V2.3 expands Uedit from being a remarkable Amiga text editor to being a powerful and flexible word processor as well. V2.3 has a Teach Mode which teaches beginners what each input does and many new features, such as tab rulers, edit-while-you-print, document/nondocument mode, and many more. Uedit can be configured (by an experienced user) to emulate popular word processors. It has most of the same capabilities they do plus far greater power, capacity, versatility, and ability to automate tedious editing chores. Uedit does not have a built-in spell-checker and doesn't show boldface and italics on the screen, but it works with any nonproportional font and with all known hardware and system modifications to your Amiga.

You can order Uedit from Rick Stiles, P.O. Box 666, Washington, IN 47501. Uedit costs $45 (US) and purchasing it makes you a registered user. You receive a serial number, $15 commissions when others register from your serialized copy of Uedit, a Shareware diskette which you can freely distribute, a private diskette with full documentation on it, notification of upgrades, custom user-written configurations such as UStar (written by Kurt Wessels) which emulates WordStar(tm) and Scribble!(tm), powerful directory utility and computer programming configurations, a number of utility programs, and a 30-day satisfaction guarantee. Uedit keeps improving because of good ideas from its hundreds of users. As a registered user you can subscribe to the Quarterly Newsletter, earn commissions for finding new users, purchase upgrades at low cost, and contribute your custom configurations and good ideas to keep this program improving.

Comspec Hard Drive for the Amiga 1000

Comspec communications has just announced the availability of their new high-performance hard drive for the Amiga. Features:

Auto-booting: in auto-boot mode, Kickstart and Workbench are loaded from the hard disk immediately after powering up; no need to keep swapping disks, even after a *Guru Meditation*. No other hard disk for the Amiga has this feature.

Able to handle media defects: Media defects are handled transparently to the user, unlike other hard drives that force you to reformat the disk.

NorthWest MUSIC CENTER INC.

539 N. Wolf Rd., Wheeling IL 60090
Hours — Phone (312) 520-2540
Mon.-Thurs. 12:30-8:30, Sat. 9:00-4:00

We want to be your Commodore shop!

Do you feel like no one cares about you and your orphaned computer! NWM cares!! We still stock and sell most major B series and 8000 series programs.

Parts available for some 8000, 9000 and B series models!

Hacker's Corner

One of a Kind • Surplus • Monthly Special • Closeouts

Limited quantities to stock on hand

8023 150cps IEEE	$179.95	Avatex 1200 modem	$94.95
4023p 100cps rehab	$99.00	9090 7.5 meg rehab	$495.00
P-I cable	$24.95	64k ram exp 8032	$125.00
I-I cable	$29.95	Monochrome Monitor	$79.95
Pet switch	$199.00	Smith Corona DM-200	$179.95
Pet daughters	$105.00		

Commodore's Superpet 9000

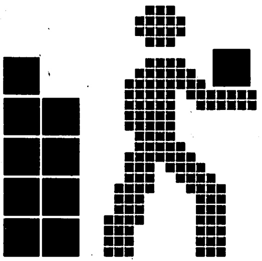
only $279.95

while supplies last

With Five Interpretive Languages:
Cobol
Pascal
Basic
Fortran
Apl

Runs 8032 software.
Great for schools and students

NWM's INVENTORY CONTROL SYSTEM*



- ■ loads program modules in less than **8 seconds** (superbase 2) to main menus in **3 seconds** or less
- ■ on screen **pop-up calculator** in transaction modules
- ■ most data centered function use the **calculator keypad**
- ■ versatile report features allow for **3 ways** to print the same report. User selects the fastest method
- ■ built in sophisticated export program allows for **complete packing** of the database
- ■ type ahead feature allowed
- ■ you can display reports on screen
- ■ access to superbase menu for user developed applications

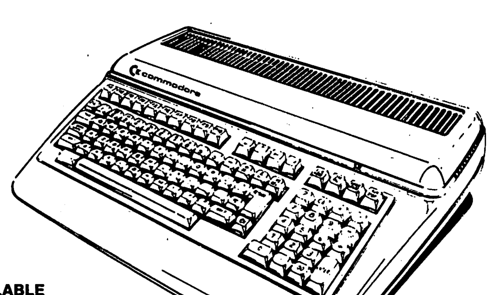
B Version 1 8050	**$49.95**
B Version 2 8050	**$49.95**
C-128 Version 1 1571	**$49.95**
B-128 Version 1&2 8050	**$54.95**

*Requires use of superbase

64K Memory Expansion for 8032 only $125 upgrades your 8032 to an 8096.

COMMODORE 8000-9000 SOFTWARE & MISC.

9000 Superpet	$279.95	Superscript 8032	$79
64K exp for 8032	$125	Superbase 8096	$79
Pet Switch	$199	OZZ Database	$25
Pet Daughters	$105	Legal Time Acc	$25
BPI General Ledger	$25	Dow Jones Program	$25
BPI Accts Payable	$25	Info Designs 8032	
BPI Job Cost	$25	Accounting System	$50
BPI Accts Receivable	$25	Superoffice 8096	$149
BPI Inventory	$25	Calc Result 8032	$89

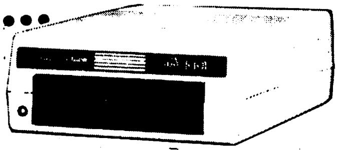


B-128 $145U.S.

NEW 128K USER INSTALLABLE MEMORY EXPANSION! INTRODUCTORY PRICE OF ONLY $125.

SFD 1001 1 Megabyte Drive double sided 8250 format IEEE interface

PRICED AT $169.95 US



SFD-1001 is the drive that you should consider when you need large amounts of data storage. It holds over 1 megabyte of data on its single floppy drive. Fast IEEE access for your C-64 or C-128. (C-64 and C-128 need an IEEE interface.) Why settle for slower drives with less storage capacity. This drive stores substantially more programs and data. Think how much money you can save on disk purchases. In fact, it stores almost 7 times more information than your standard drive. Bulletin board owners love them. And what an introductory price! At $169.95 these drives will sell fast, so don't wait. This drive has the identical format of a CBM 8250 drive, one of Commodores most durable floppy drives.

MODEL	SFD-1001	Sector/Cylinder	—
DRIVES	1	Sector/Track	23-29
HEADS/DRIVE	2	Bytes/Sector	256
STORAGE CAPACITY (Per Unit)		Free Blocks	4133
Formatted	1.06 Mb	TRANSFER RATES (Bytes/Sec)	
MAXIMUM (Each Drive)		Internal	40 Kb
Sequential File	1.05 Mb	IEEE-488 Bus	1.2 Kb
Relative File	1.04 Mb	ACCESS TIME (Milli-seconds)	
Disk System		Track-to-track	*
Buffer RAM (Bytes)	4K	Average track	**
DISK FORMATS (Each Drive)		Average Latency	100
Cylinders (Tracks)	(77)	Speed (RPM)	300

SOFTWARE FOR THE B-128!!!

		C.A.B.S. Accounting	
Superbase	$19.95	General Ledger	$ 9.95
Superscript	$19.95	Accounts Receivable	$ 9.95
Superoffice Integrated		Accounts Payable	$ 9.95
Superbase & Superscript	$49.95	Order Entry	$ 9.95
Calc Result	$89.95	Payroll	$ 9.95
Word Result	$89.95	Buy all 5 for only $24.95	
Super Disk Doc	24.95	Superbase: The Book	$14.95
The Power of: Calc Result (Book)	$14.95	Applied Calc Result (Book)	$14.95

ORDER NOW WHILE STOCK LASTS!

Send or call your orders to: **Northwest Music Center, Inc. 539 N. Wolf Rd., Wheeling IL 60090. 312-520-2540** For prepaid orders add 16.90 for 8023p, $25 Superpet, 9.95 SFD 1001, 10.95 B-128. 9.95 4023p, 15.95 9090 and 4.95 64K memory expansion. For software add $3.00 for first and $1.50 for each additional book or program. Canadian shipping charges are double U.S. International orders, call for shipping. For C.O.D. orders add 1.90 per box shipped. All orders must be paid in U.S. funds. Include phone numbers with area codes. Do not use P.O. Box, only UPS shippable addresses. A 2 week hold will be imposed on all orders placed with a personal or business check. C.O.D. orders shipped in U.S. only and cash on delivery, no checks. 30 day warranty on all products from NWM, Inc. No manufacturer warranty. NWM reserves the right to limit quantities to stock on hand and adjust prices without notice!

All prices quoted in US dollars.

Intelligent SCSI Controller: Allows for high-speed multitasking, leaving the Amiga free to do other tasks while the drive seeks and retrieves data; other hard drives will not even allow you to enter keystrokes while these operations are occurring. The SCSI controller also allows for maximum expandability, allowing the user to add additional hard drives, tape streamers, etc. The SCSI controller also contains a battery backed-up clock, which automatically sets the system time on power-up.

Hard Drive Chassis: Allows enough room to add hard drives from 10 to 300 megabytes, or space to add a second half-height hard drive or tape streamer. It comes with a power supply that is capable of running on voltages from 100 VAC to 240 VAC, at 50HZ or 60HZ without setting any switches or jumpers. The chassis is equipped with a low noise fan to protect from overheating and provide greater reliability.

The drive can be ordered with one or two 20 or 40 megabyte drives in the chassis; larger sizes are also available.

For pricing and other information, contact: Comspec Communications Inc., 153 Bridgeland Ave, Unit 5, Toronto, Ont. M6A 2Y6, Phone: (416) 785-3555 Fax: (416) 785-3668.

ROMDISK with HYPERBOOT for the C64

Epimetheus Corporation has introduced its 128K-byte ROMDISK with HYPERBOOT for the C64 and C128 in 64 mode. It combines all the hardware and software you need to create a library of up to 150 of your favourite programs on an EPROM bank attached to the user port. Transferring program files from a 1541 disk drive to the ROMDISK is made simple with a menu-driven program. This software, called HYPERBOOT, is provided on an 8K cartridge. Once programs are transferred from floppy disk to the ROMDISK, they load at a rate of 16,000 bytes per second. The ROMDISK comes in a finished case with all 128K bytes of EPROM installed. It is erasable using ultraviolet light and can be reprogrammed thousands of times. Also available are two-way switches that allow both a modem and the ROMDISK to occupy the user port ($39 US) and a 3-foot extension ribbon cable to allow remote placement of the ROMDISK or a modem ($24.95 US). ROMDISK with HYPERBOOT sells for $179.00 (US).

Contact: Epimetheus Corporation, P.O. Box 171, Clear Creek, Indiana 47426 (812) 336-4508.

MIDI Interface for C64/128 and 64C

Audio Digital Processing Systems is announcing the release of MIDI 64, an intelligent MIDI interface for the C64/128 and 64C computers and all MIDI-equipped instruments and devices.

MIDI 64 is the only all-Canadian MIDI interface, and introduces many innovations, such as a 16K auto-boot bank-switched EPROM containing: an extensive MIDI supplement to BASIC for easy custom MIDI programming; a real-time 4-track sequencer; a MIDI-data monitor for hex/binary/decimal real-time data display; an interface/cable auto-test program. All programs are automatically available on power-up, and can be switched out to run other software.

The MIDI 64 package includes: MIDI interface with EPROM installed, two MIDI cables, full documentation on disk, and MIDI BASIC program examples.

With a suggested Canadian retail price of $199.95, ADPS is targeting MIDI 64 at musicians, home hobbyists, students, repair technicians, and small recording studios. The package is especially well-suited for programming and MIDI teaching in the classroom, and first-time MIDI users. Dealer and distributor inquiries are invited. Contact: ADPS, c/o Phil Honsinger, 86 Foxhunt Road, Waterloo, Ontario, N2K 2Z6, (519) 886-6361.

GEOS Upgrade for the C128

Berkeley Softworks is offering GEOS for the C128 to registered C64 GEOS owners for just $22.00 (US) plus $2.40 shipping/handling. The 128 version of GEOS runs at 2 MHZ, supports 80-column mode, and uses the optional 1750 RAM expansion unit as a RAM-disk. Contact: Berkeley Softworks, 2150 Shattuck Avenue, Berkeley, California, 94704, (415) 644-0883.

Video Digitizer for the C64

Eye-Scan from Digital Engineering is a video digitizer for the Commodore 64, 64C, SX64 and C128. Eye-Scan's cartridge plugs into the user port, and accepts a composite video signal via a standard RCA jack. Conversion time is approximately six seconds per gray level. Eye-Scan's software uses pull-down menus and allows black and white imaging, up to eight gray levels, image inversion, and 1525 printer support. Also included is a programmer's utility package that allows programmers to use the imaging capturing algorithms in their own programs. Possible applications include animation, security, automated process control, pattern analysis, robot vision, and text recognition. Contact: Digital Engineering and Design, 2718 S.W. Kelly Suite C165, Portland, Oregon, 97201, (503) 245-1503.

C128 CP/M Kit

INCA announces the release of a CP/M kit for the Commodore 128. The kit consists of a 39-page booklet and two disks of CP/M public domain software which: explains the use of some of the programs on Commodore's system disk; demonstrates some of the main features of Commodore's CP/M; shows how to get started using a modem to obtain more CP/M programs and information.

The "CP/M kit for the Commodore 128" lists for $29.95 (US). Contact: INCA, 1249 Downing St, P.O. Box 789, Imperial Beach, CA 92032.

Aegis Releases "Art Paks" for the Amiga

Aegis Development, Inc. has released "Art Pak, Volume I" for the Amiga, consisting of art done by Aegis' professional art team. Volume 1 includes photograph-quality artwork of buildings for use as backdrops, pieces of cel animations for creating one's own walking, moving animations. Since Aegis Animator can do both metamorphic and cel animation, these images can be used with both styles as different things.

Art Paks can be used with Aegis Images professional paint program, Aegis Animator, and Aegis Draw, the entry-level CAD program for the Amiga. Any program that can read IFF format graphics files will be able to take advantage of these images. Art Paks will sell for $34.95 (US) at retail outlets for Amiga software.

Genealogy Program for the 8032

Byteware now has a version of "the Genealogist" for the PET 8032, following their versions for the 64, 128 and Plus/4. This program vastly eases the Genealogist's record keeping tasks, and is available in 4040 or 8050 format. An Amiga version is under development. Send a SASE to the following address for information, sample sheets, and prices for the various programs (prices start at a very reasonable $9.95 US). Maple City Software, 906 West 6th Ave., Monmouth, IL, 61462.

8051 Cross Assembler for the C64 and C128

The Zipp-Code-51 is an assembler for the 8051 processor, used in smart peripherals, robotics and other applications. The editor used is the standard BASIC editor of the 64 or 128. There are two versions of Zipp-Code-51, optimized for either the 64 or the 128. The package also includes a symbol cross reference utility, a disassembler, and a binary to source file converter. Price is $49.95 (including shipping); specify C64 or C128 when ordering. Order from: Hughes Associates Software, 45341 Harmony Lane, Belleville, MI, 48111, (313) 699-1931.

Late-BRKing News: Oxxi Claims Benchmark M2

This just in, as they say. Last issue we ran a press release for the Benchmark Modula-2 development package. The source for the software was given as Oxxi Inc. of Fullerton, California, the company from whom we obtained the pre-release copy briefly reviewed in this *Transactor* (see page 59). A few weeks back, we were informed by Leon Frenkel, author of Benchmark, that he had severed relations with Oxxi and would henceforth be marketing his Modula-2 through his own newly-formed company, Avant-Garde Software. Accordingly, it is Avant-Garde's name (and pricing) that appears with the review. Just as we are about to go to press, however, we have been told by John Houston of Oxxi that his company is disputing Frenkel's claim to have title to Benchmark. Well, we aren't lawyers, and we don't know which way this particular cookie will crumble. We *can* tell you that Benchmark is (at this instant) available from Avant-Garde only, that the question of ownership will probably be resolved one way or another by the time we print our next issue, and that whatever happens the product does exist and will be supported. By someone.

THE TIME SAVER

Type in a lot of Transactor programs?
Does the above time and appearance of the sky look familiar?
With The Transactor Disk, any program is just a LOAD away!

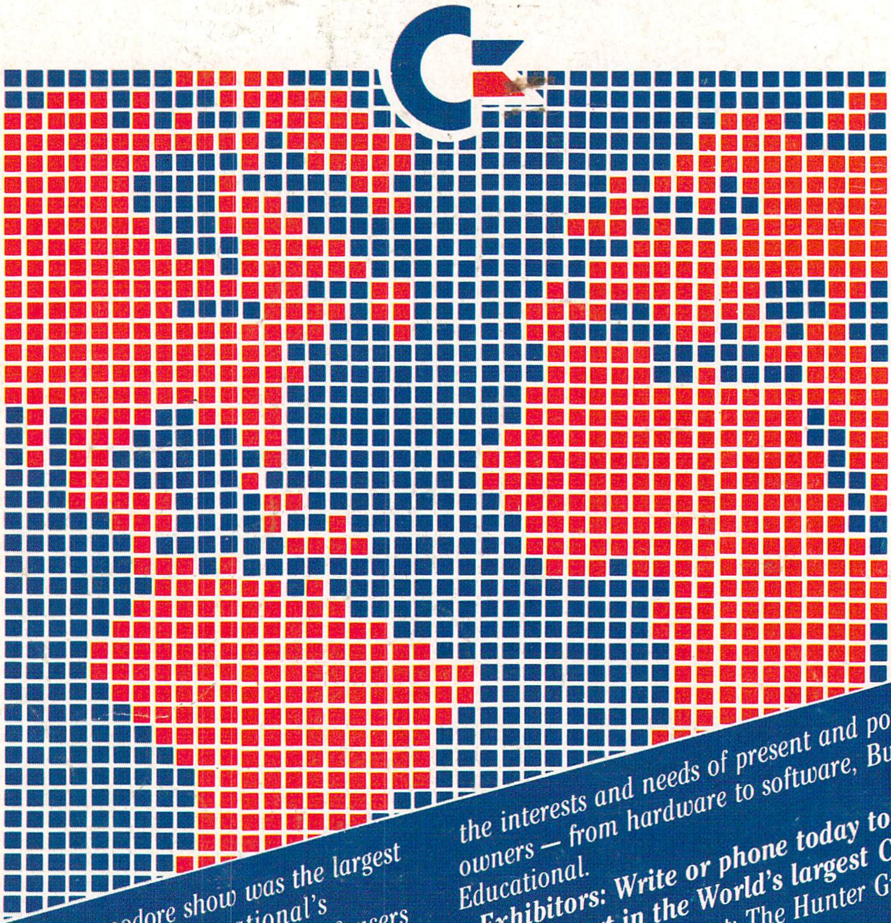
Only $8.95 US, $9.95 Cdn. Per Issue
6 Disk Subscription (one year)
Just $45.00 US, $55.00 Cdn.
(see order form at center fold)

Now Amiga Owners Can Save Time Too!

Transactor Amiga Disk #1, $12.95 US, $14.95 Cdn.

All the Amiga programs from the magazine, with complete documentation on disk, plus our pick of the public domain!

THE WORLD OF COMMODORE



The 1986 Canadian World Of Commodore show was the largest and best attended show in Commodore International's history. With 350 booths and attendance of over 38,000 users it was larger than any other Commodore show in the World — and this year's show will be even larger.

World of Commodore is designed specifically to appeal to the interests and needs of present and potential Commodore owners — from hardware to software, Business to Personal to Educational.

Exhibitors: Write or phone today to find out how you can take part in the World's largest Commodore Show.

For information contact: The Hunter Group Inc. (416) 595-5906

INTERNATIONAL CENTRE
DECEMBER 3-6, '87
TORONTO