TIsHUG NEWS DIGEST

Focusing on the TI99/4A Home Computer

Volume 11, Number 5 June, 1992

Registered by Australia Post - Publication No. NBH5933

Memo

To: Gary Bowser of OPA

From: TIsHUG (Australia) Limited

Subject: Fifteen TIM and SOB cards Paid for in October 1991, Still waiting in June 1992.

We are tired of waiting!

Sydney, New South Wales, Australia $3

TIsHUG (Australia) Ltd
A.C.N. 003 374 383

TIsHUG News Digest

June 1992

All correspondence to:

P.O. Box 1089
Strawberry Hills, NSW 2012
Australia

The Board

Co-ordinator
Dick Warburton (02) 918 8132
Secretary
Terry Phillips (02) 797 6313
Treasurer
Geoff Trott (042) 29 6629
Directors
Rolf Schreiber (042) 85 5519
Russell Welham (043) 92 4000

Sub-committees

News Digest Editor
Bob Relyea (046) 57 1253
BBS Sysop
Ross Mudie (02) 456 2122
BBS telephone number (02) 456 4606
Merchandising
Percy Harrison (02) 808 3181
Publications Library
Russell Welham (043) 92 4000
Software library
Rolf Schreiber (042) 85 5519
Technical co-ordinator
Geoff Trott (042) 29 6629
TI-Faire co-ordinator
Dick Warburton (02) 918 8132

Regional Group Contacts

Central Coast
Russell Welham (043) 92 4000
Coffs Harbour
Kevin Cox (066) 53 2649
Glebe
Mike Slattery (02) 692 8162
Hunter Valley
Geoff Phillips (049) 42 8176
Illawarra
Geoff Trott (042) 29 6629
Liverpool
Larry Saunders (02) 644 7377
Northern Suburbs
Dennis Norman (02) 452 3920
Sutherland
Peter Young (02) 528 8775

Membership and Subscriptions

Annual Family Dues	$35.00
Associate membership	$10.00
Overseas Airmail Dues	A$65.00
Overseas Surface Mail Dues	A$50.00

TIsHUG Sydney Meeting

The June Meeting will start at 2.00 pm on 6th of June at Ryde Infant School, Tucker Street, Ryde. The Assembly Class will run from 10.00 am to 1.00 pm for those wishing to attend.

Printed by
The University of Wollongong
Printery Services.

TIsHUG News Digest **ISSN 0819-1984**

Index

All articles appearing in this month's issue of the TND are available as text files on disk ready for the formatter. Newsletter editors please note that, if you wish to re-print any articles, contact us, stating which articles you are interested in and giving the date of the TND. These will be dispatched to you promptly at the cost of the media plus postage.

Editor's Comment

by Bob Relyea

I sure was not disappointed at the tutorial last month. I attended a well-structured session on TIPS by Alf and learned a lot, including information about his update version. As usual, in the process of working through the tutorial, we got side tracked with various questions and I picked up a few pointers that will help me a lot. For example, it was pointed out that it is possible to print a catalog of a disk directly from the MENU of my Ramdisk Menu. I also learned how to change the parameters so you can do this with either PIO or RS232. Great! It was also nice to see so many there including a couple of members of note that we had not seen in a year or more. Have fun reading through the magazine.

Co-ordinator's Report

by Dick Warburton

Recently I examined a new notebook computer. I was quite impressed with its power, versatility and capacity. It included a 40 megabyte hard drive, would run on its batteries for long periods and process large complex programs very quickly. When you allowed for the hard drive, the floppy drive, the keyboard, the LCD screen and batteries, it was hard to imagine where they could actually put the circuitry for the computer. It set me thinking about the rapid technological development which has occurred over the past few years. The rate of development has been phenomenal. One can only wonder what an up to date TI99/4A would be like today if it had been supported and developed with the latest technology.

I still find that I have only mastered only a small fraction of what is available for the TI99/4A. There is still so much to learn and do with this machine. Compared with the latest developments, it is prehistoric. However you can actually learn quite a lot about it, because it is accessible, open and basically understandable. I suspect that few notebook users will ever know anything more about their machines than how to run a few selected programs. Certainly they will not attempt to get inside the computer. The more modern computers will leave people quite ignorant about how they work and about basic principles, because they are generally inaccessible. The technology has been minaturized. Parts now are too small to fix. I suspect that with mass production, whole computers will become throw away items in time. They will simply not be repairable.

With such technology comes great benefits, if we are skilled enough and able to exploit its use for our advantage. If we can master the software and can see the applications, we have fantastic tools to work with. Most people, I fear, like me, are not brilliant, learn fairly slowly, tend to stick to things we know well and lack the imagination and skills to make the most of using them successfully. I suspect that for most people, computers will remain glorified typewriters, which can be switched over to playing games, or storing some data from time to time. I fail to see how the majority of computer users will benefit from the latest computer designs. They will pay big bucks for the latest and fastest machine. I know that they can carry their computer wherever they go, perhaps even play noughts and crosses at the traffic lights. They can impress their friends with the speed and smoothness of operation. Some may even use them as high priced typewriters. Just imagine every police officer being issued with latest notebook computer. They could take their statements on the spot, typing them in as we speak (one finger only of course). Imagine the possible savings in time and money. Bus drivers could record all sorts of data about passengers while collecting the fares. A new arm of the Department of Census and Statistics could be created. It could open up a whole new way to collect Government information. However, for most users, the usage will be fairly simple and uncomplicated.

The problem I see developing, is that many people will become dependent on these mind tools, unable to understand how they work and in time unable to use the basic skills to solve their own problems. They will become even more dependent on technology and very dependent on the suppliers of hardware and software, unable to modify or repair the systems they use. They will have little computer knowledge beyond their own area of expertise. However, TI99/4A users who get really interested, can gain so much more from this simpler but versatile computer. TI99/4A users can select word processing or data management if they choose, or they can load and play fascinating games, or challenge themselves with chess or solitaire. TI99/4A users have a computer which uses speech quite well, can program surprisingly good music, can be used by the artistic to create great designs and pictures and can display much of the best artwork available on other machines. The list goes on and on: control a model train system or house alarms; use a spreadsheet; write books; do complex engineering computations. I never cease to be amazed at what a humble TI99/4A can achieve in the right hands. Wait till you see Ben's library program in action; the Mudie revision. The use even of a TI99/4A is limited only by our imagination and intelligence. Features we take for granted on a TI99/4A are often costly additions on another system, for example speech. Use a RAMdisk and your system will be up and running faster than the latest IBM clone. Some of our members can either write or modify the programs we use. We can modify or develop our own hardware. We can repair our own systems, cheaply. TI99/4A users can learn how things work more easily. TI99/4A users can have great enjoyment doing a whole range of things with their TI99/4A. Other computers all have features in some areas which are vastly superior to a TI99/4A.

e.g. Amount of memory for programs.
Vivid graphics.
Stereo Sound.
Superb desk top publishing.

However, I do not know another machine, on which I can do so many things. I use a 286 machine at times. The quality of its sound makes me squirm. It takes 45 seconds to load its basic system and get ready to run. DOS is convoluted and awkward to use etc. My wife's Amiga system has brilliant graphics and sound. It multitasks from a windows environment. However I get fed up with the time it take to load each time it is used. Getting it repaired when it broke down was a nightmare, even under warranty. The problems went on for months and no-one seemed to know how to fix it. We finally received a new main board, which still played up. While I have access at any time to a sophisticated Amiga system, I find my simple but versatile TI99/4A to be more satisfying and enjoyable.

You can get more from your TI99/4A. You can expand it further. The club can still help you to do it quite cheaply. Disks and software are cheap. You can save money with standard equipment. Unlike the Macintosh, you are not locked into expensive peripherals and parts. You can learn to build a RAMdisk, fix a console, repair a keyboard or power supply, make your own PIO/RS232 cables, even learn about using EPROMs on RAMdisks. You can learn to program in a number of languages, even learn how to fix monitors. Most of the technology in use today, is understandable if we understand how a TI99/4A works. There are developments I would like to see for the TI99/4A. I am waiting for an interface for a CD ROM player (with software). I would also like to see a portable expanded system, one I could carry around easily. I would love to see an interface for standard hard disk drives, other than the Myarc. I suspect that all these goals will be achieved in the near future, because of the challenge the TI99/4A offers to so many people world wide. The TI99/4A has brought me tremendous enjoyment and satisfaction, let alone friendship and company with a great group of people. The TI99/4A has expanded my horizons in so many ways I could hardly believe possible. I still think it is a remarkable machine, when we consider how long ago it was conceived. I suspect it will become the Rolls Royce of vintage computers.

Do not forget the Faire. Keep those ideas rolling. See Ian Mullins about how you can help. o

Treasurer's Report

by Geoff Trott

The renewals have been coming in and so we have had a good month at the bank. Now if we can just obtain taxation exemption status and become a non-profit company in the eyes of the Australian Securities Commission and we will be in good shape for our old age.

Income for April	$2935.40
Payment for April	$608.82
Excess of income over expenses for April	$2326.58 o

Secretary's Notebook

by Terry Phillips

Hope you made it along to the May meeting where a very interesting and informative day was had by all. There were plenty of demonstrations; we seem to have a particularly high interest in Page Pro, with members huddled over the computer set up for hands on experience. There was plenty of interest in hardware, particularly getting RAMdisks up and running plus games and adventuring. In fact, there was something for everyone, including a great BBQ lunch of sausages with all the trimmings. I think everyone had their fill. I know I was pretty exhausted after slaving over the hot stove for a couple of hours.

Memberships are continuing to roll-in, but fairly SLOWLY. At the time of writing this article there are 44 members, with membership expiry dates of 30 April, who have yet to renew. If you know of someone who has not as yet renewed, please have a word in their ear. I will have a list of those people concerned at the next meeting, so if you want to you can have a look to see if any of your acquaintances are on it.

Three new members have joined us since last month and it is a big welcome to:

Mick Blayney	Saratoga
Lloyd Gibson	Thallon (Qld)
Daryl Boon	Tahmoor

Thallon, by the way, appears on my road atlas as a small town about 70km over the New South Wales border, near Mungindi. I doubt whether you could get a more out of the way place for a member of the club; never the less, Lloyd, if you run into any computing problems give us a call and we will do our best to solve them.

Good to see Neal Negal from the Central Coast at the May meeting. Neal, looking a picture of health, wrote me a letter recently as follows:

"We have a monthly magazine at the retirement village and I was able to print out Christmas & New Year good wishes with pictures of deers, bells and a very thin Father Christmas. Very gradually some of the functions of the computer and the printer are entering my grey matter, but I find it great fun. My wife thinks it incredible how I can print pictures. My greatest frustration is fully understanding why, when I press what I think are the right keys, the answer is a LEMON. It has taken me 30 minutes to print this letter and I could have written it in half that time."

Keep at it Neal. You may have been able to write the letter in half the time, but I am sure you had more fun doing it on the computer rather than with pen and paper!

Jack Strong from Collaroy sent a short note with his renewal advising that he has been caravaning for the past couple of years, but hopes a more ordered life will enable him to attend meetings this year.

With his renewal, Milton Mable of Malabar sent a note thanking Geoff for the work he has done on his RAMdisk, which is now up and running.

John Last from Mortdale, with his renewal, advised that his TI99/4A is doing service with his 78 year old father who is a project officer with the Volunter Coast Guard in Wollongong.

That's all for this month. Do not forget to come along and vote on the proposed changes to the club's Memorandum and Articles of Association at the June meeting. Full details of this appeared in the May newsdigest.

Renew Now!

Decoding EPROM files

Appendix, by Ben Takach

The edited EPROM code, the basis of the decoding program. Information stored on EPROMs is always in such format.

This is the last in the series.

```
7FFF AFFF BFFF 9F03 9F06 9F31 9F6A 9F38 9F39 9F4C 9F6B 9230 9234 8C07 9F73 9F74
0000 0000 0000 0000 0000 7FFF 9F40 9F43 9F44 9F45 8F48 9F49 9F4B 9F50 9F5E 9F61
9F62 9F64 9F65 9F66 9F67 9F68 9227 9F6E 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 1000 3F56 8FFE 1043 4024 5FFE 5F06 3F69 4235 4217 8F00
0000 0000 2000 3F56 8FFE 1044 4024 5FFE 3F69 4235 4F06 4217 8F01 0000 0000 2020
5F03 3F06 8F03 0000 0000 1000 3004 3006 4F0A 4045 8F04 0000 0000 1020 3F05 5001
3F06 8F05 0000 0000 2020 5F06 3F55 3024 8F06 0000 0000 1043 5044 4F06 5F03 8F07
0000 0000 1054 3F69 4052 8F08 0000 2054 3F69 8F09 0000 1054 4052 3003 8FFE 1F09
3002 5FFE 8F0A 0000 0000 104C 403F 8FFE 1056 4045 5FFE 5024 503F 5045 8F33 0000
0000 0000 1056 0000 8F69 0000 0000 1F38 6057 3F0A 3006 3004 4F4D 8F10 0000 0000
1F38 6057 3002 3006 3005 8F11 0F11 0000 1003 3006 3001 8F12 0000 0000 1003 3007
3000 8F13 0000 0000 1000 3005 3007 8FFE 1F09 3001 3004 3006 5FFE 8F14 0000 0000
1001 3005 3006 8FFE 1F09 3000 3004 3007 5FFE 8F15 0000 2045 5F30 3029 4F6D 500B
604C 3000 3005 3002 3F1A 8F16 0000 1029 500B 604C 3F0A 3004 3000 3F1A 8F17 0000
0000 1F25 3002 3005 3000 3006 8FFE 1002 3005 3001 3006 3052 4053 400A 5FFE 6057
8F18 0000 1052 3008 6052 5000 8F76 0000 1000 3004 300A 5F73 3052 4001 4009 4F53
8F73 1052 300A 6052 8F77 0000 0000 1F73 400A 4052 4053 8F74 0000 1F0A 3004 3000
3006 8FFE 1F0A 3004 3001 3006 5FFE 6057 8F19 1F39 6057 8F1A 0000 0000 2048 502D
3028 302A 5F30 6F03 303F 4F13 8F30 0000 0000 1001 3F06 5F4C 6F03 4026 8F4C 0000
0000 0000 1001 3F06 5F4D 3F03 4026 8F4D 0000 0000 1F00 3F10 8FFE 1F01 3F11 5FFE
3F69 3020 4F05 4F27 3F77 8F26 0000 1F00 3F11 8FFE 1F01 3F10 5FFE 3F69 4F26 3F77
8F27 0000 2055 5023 3F00 3F12 8FFE 1F01 3F13 5FFE 3F69 4F29 8F28 0000 0000 1F56
3000 8FFE 1F00 3F13 5FFE 8FFE 1F01 3F12 5FFE 3F69 4F28 8F29 0000 0000 1022 5055
5000 3F00 3F15 8FFE 1F01 3F14 5FFE 8FFE 1F04 4054 5FFE 3F69 4F2B 3F76 8F2A 0000
1F00 3F14 8FFE 1F01 3F15 5FFE 8FFE 1F04 3054 5FFE 3F69 4F2A 3F76 8F2B 0000 1F25
5057 3F00 3F16 8FFE 1F01 3F17 5FFE 3F69 4F56 4F2D 8F2C 0000 0000 1F00 3F17 8FFE
1F01 3F16 5FFE 3F69 4F2C 8F2D 0000 0000 1F00 3F19 8FFE 1F01 3F18 5FFE 6057 3F69
4F2F 8F2E 0000 2F06 5F25 5001 3F00 3F18 8FFE 1F01 3F19 5FFE 3F69 4F2E 3057 8F2F
0000 0000 1F00 5F01 3206 3007 5F6B 3000 4006 8F6B 0000 0000 1F26 3F69 3020 3F07
3006 8201 4001 8225 0000 0000 1F27 3F69 3006 8200 4000 8224 0000 0000 1F28 3F69
4005 8205 0000 0000 1F29 3F69 4004 8204 0000 0000 1F2A 3F69 4002 8202 0000 0000
1F2B 3F69 4003 4F6B 8203 0000 0000 1000 3F2C 3F69 4007 4F6B 8207 0000 0000 1F2D
3F69 4006 8206 0000 0000 1F2E 4053 8FFE 1F2F 6057 3053 5FFE 4008 3F69 4F73 820B
4008 820F 0000 0000 1F2E 3053 8FFE 1F2F 4053 5FFE 8FFE 1009 320A 420B 5FFE 3F69
3057 5F74 820A 4009 820E 0000 0000 1208 4F00 4045 8FFE 2023 3022 5FFE 4055 3001
8208 0000 0000 1F06 3F69 8209 0000 0000 1200 4000 8FFE 1201 4001 5FFE 8FFE 120B
520A 3057 5FFE 3F69 504C 8222 0000 0000 1211 5210 5215 5F63 5216 5226 5239 523A
820C 0000 0000 0000 2C07 5211 5210 5215 6238 8235 0000 0000 1055 5005 3001 3F03
8FFE 2045 3050 3056 5FFE 8FFE 2233 3024 5FFE 8231 0000 0000 0000 0000 0000 0000
1001 3005 3002 5232 4000 3056 8232 0000 0000 0000 0000 1F25 5230 3000 3F4C 8230
0000 0000 1230 5234 3F06 3020 3F03 8234 0000 0000 0000 1056 8233 0000 0000 2021
6051 3056 8F54 0000 0000 2051 5F54 8217 0000 0000 1021 4F54 8236 0000 0000 2015
501D 4014 501C 4013 501B 4012 501A 4011 5019 4010 8F31 0000 0000 2F31 3001 5C01
4018 8FFE 1024 3F47 5FFE 8F47 0000 0000 1F47 3000 8211 0000 0000 1005 3002 3006
3000 3F06 4217 403E 8C01 0000 0000 1015 501D 3014 501C 3013 501B 3012 501A 3011
5019 3010 5018 8F24 0000 0000 1F6A 4211 6024 8F25 0000 0000 1F24 5F6A 503E 4007
3024 8F6A 0000 0000 0000 1001 3000 8FFE 1002 3003 5FFE 8FFE 1007 3006 5FFE 8FFE
1008 3009 5FFE 5C00 5210 600F 3F33 3056 8210 0000 0000 1005 3004 8C00 1008 4053
8FFE 1009 3053 5FFE 8FFE 1052 3F77 5FFE 8F38 1008 3053 8FFE 1009 4053 5FFE 8F39
0000 0000 0000 1035 305D 304C 4217 8223 0000 0000 0000 0000 0000 0000 0000 0000
2058 6059 8239 0000 0000 105A 505B 823A 0000 0000 1058 405A 8238 0000 0000 2035
EFFF C015 1054 3003 8FFE 2054 3002 5FFE 3000 3004 3006 3F72 8F58 100A 6052 8F72
2048 502E 302C 3029 8F59 1F53 4F55 4F58 3F69 8F56 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 1F56 503E 4043 4044 3F52 4040 4F55 8F53 0000 0000 2042 4041
4039 4038 403C 403D 8F52 0000 1F55 3004 3003 3000 5F50 4F71 304A 8F50 8C03 0000
1029 500B 303A 5F50 304A 4C03 5F6D 8218 0000 0000 104A 4218 8219 0000 0000 121B
304A 3F68 303F 3029 8C08 8F6C 0000 0000 1F6C 4C08 8F6D 102A 3028 5F51 4C04 8F51
0000 0000 1F3E 521D 3F55 3F51 8C09 8F3E 0000 0000 1C09 503B 3049 8221 8C04 0000
0000 103C 303D 402A 3F40 8FFE 103C 403D 5FFE 8FFE 203C 303D 5FFE 5C0A 4F3C 304C
8C0A 0000 0000 1C0A 8226 0000 0000 1F3F 303F 8FFE 1006 403F 3041 5FFE 302C 3F5E
8F36 0000 0000 0000 0000 0000 0000 0000 0000 1C06 303F 4F62 400B 8FFE 1042 5F57
403F 5FFE 302C 3006 8F37 0000 100B 3028 8F79 0000 0000 0000 1F3C 3F4B 302C 8F3D
0000 0000 1F3A 4F69 4C0A 4C06 8FFE 1C03 604A 3F3D 5FFE 302C 8F3F 0000 0000 0000
2041 4042 403F 4F57 8F34 0000 0000 1F40 302C 3F55 4029 8C06 0000 0000 1F36 521A
8FFE 1028 421B 6F34 3FFE 4F37 402B 3F3B 5F79 821A 0000 121A 4028 5F79 822D 0000
1F37 521B 8FFE 1029 421A 6F34 3FFE 4F36 4213 3F3B 400B 821B 0000 121B 4029 822C
0000 0000 1007 5F43 4028 3045 8F43 0000 0000 1006 3004 3F43 5F4B 4028 3045 8F4B
0000 0000 0000 104C 305D 500B 4217 4215 3035 0000 8F3B 0000 0000 0000 0000 0000
0000 0000 1F58 3F55 303C 303D 4F69 3F3B 8FFE 100B 3028 303C 303D 3F3B 5FFE 8F3A
0000 0000 0000 0000 0000 0000 0000 1F55 3F69 3F3B 400B 8F3C 0000 103F 304C 8227
0000 0000 1045 6F69 8F5D 0000 103F 500B 3F5D 8FFE 1F5D 404C 5FFE 4F5A 8F55 0000
0000 2F69 404C 8F5A 0000 0000 203F 300E 8FFE 123F 5F5C 3028 303F 4F48 5FFE 4F4A
300B 3051 4F32 823F 0000 0000 0000 123F 300B 305E 5F4A 423E 8F4A 0000 0000 0000
0000 0000 203F 300D 5F32 8FFE 421C 300B 3051 823E 100B 305D 405C 303F 402A 4228
3F75 4F32 8F78 0000 0000 1228 5F32 405C 300B 8F32 100B 305C 600B 5F75 303F 8F75
103C 303D 5F3C 3F5E 8FFE 1F5C 302A 5FFE 3028 4C0A 4F46 5F78 8F5C 0000 0000 0000
104D 3F5C 304E 5F6E 4F70 304D 8F6E 0000 0000 0000 0000 0000 1F44 302E 304D 8F70
0000 0000 0000 0000 0000 0000 0000 1F48 402C 4F5C 421C 304C 3C0C 8FFE 2028 402C
304C 0000 5FFE 821D 402C 8228 0000 0000 0000 0000 1F44 302E 6048 8F60 0000 0000
102A 5034 3F60 5F46 4029 405C 8F46 0000 1C0B 6F55 3F5C 5F5B 421D 4F48 3F3B 4F63
3037 4F6E 8FFE 105D 405C 3028 3FFE 821C 402A 8229 0000 0000 0000 0000 1040 403F
8FFE 103F 3F46 5FFE 3028 0000 3F5E 8FFE 1036 3F48 5FFE 8FFE 120C 0000 4217 5FFE
4F5C 8F48 0000 0000 0000 0000 1048 504D 302D 5F44 402C 8F44 0000 0000 2F55 302A
3F44 404D 8F5B 0000 0000 102A 5F40 4029 303F 8F40 0000 0000 102B 321C 5F49 402A
8F49 0000 0000 1F5C 402A 4F49 8F4E 0000 0000 1F48 402C 8F4F 0000 0000 1F4E 303F
8FFE 1F4F 3F55 5FFE 3F3B 8FFE 2028 402C 304C 305D 5FFE 303F 821E 0000 0000 0000
1F5C 3F3B 5F49 3F55 8C0B 0000 0000 1F48 304C 3F55 603F 6045 8C0C 0000 0000 1048
504D 8F6F 0000 0000 0000 0000 102E 6F6F 3F3B 8F5E 0000 0000 102A 3048 3034 8FFE
104D 3F6E 5FFE 8F5F 303F 4F44 8C05 0000 0000 0000 0000 0000 0000 1C05 303F
8FFE 1038 403F 5FFE 8FFE 1F5F 502C 3FFE 3F6F 8F41 0000 0000 1F41 521F 8FFE 102D
4220 6F35 3FFE 4F42 3F6F 3F3B 821F 0000 0000 121F 402D 822A 0000 0000 1F44 303F
8FFE 1039 5F57 403F 5FFE 3F6F 8F42 0000 0000 1F42 6F6F 5220 8FFE 102E 421F 6F35
3FFE 4F41 3F3B 8220 0000 1048 504D 3220 402E 822B 0000 0000 0000 0000 2038 4039
403F 4F57 8F35 0000 0000 104C 405D 400B 5F4A 8216 102A 302C 8FFE 1029 3028 5FFE
8FFE 102E 302D 5FFE 8FFE 105C 305D 5FFE 5215 3F33 8215 1032 5F61 422D 8F61 0000
0000 1033 5F64 422D 8F64 0000 0000 1F4F 5F62 5F49 8FFE 2F61 6F64 3FFE 3F68 403E
304B 303F 8F62 0000 0000 1C06 3F62 3028 5C0D 303F 5213 403E 8213 0000 0000 0000
1032 5033 304B 8C0D 0000 0000 0000 1213 5214 8F63 0000 0000 1030 5F65 402B 8F65
0000 0000 1031 5F67 402B 8F67 0000 0000 1028 302A 5F68 303F 8F68 0000 0000 122F
5F66 8FFE 2F65 6F67 3FFE 3F68 403E 304B 303F 8F66 0000 0000 0000 1F66 3028 5C0E
303F 5214 403E 8214 0000 0000 0000 1030 5031 304B 8C0E 0000 0000 0000 203F 3047
5F45 302C 3029 8FFE 122F 303F 5FFE 421C 4F5C 3F3B 822F 0000 0000 121B 5F45 4F5C
303F 8F45 0000 0000 102C 3046 403F 421C 3F3B 822E 0000 0000 200B 3028 8FFE 100B
3001 5FFE 8F71 0000 0000 0000 0000 C035 FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
```

TIM = TI-Image Maker

by Christopher Pratt, PA USA

TI IMAGE MAKER is an internal console expansion board designed to be installed in your 99/4A with a few simple tools. TIM upgrades your current TMs9918 video display processor to the latest state-of-the-art and fully compatible V9958 processor. This processor is also compatible with the V9938 used in other upgrades. OPA (Gary Bowsers company) used state-of-the-art CAD/CAE designing and top design engineers to bring you the best video upgrade for the TI in the smallest most compatible package possible. The PC Board containing the V9958, 192K of VRAM, a special ASIC device, a 25-pin monitor/expansion port for future video devices like digitizers, GENlock, etc. and the analog RGB video driver circuitry, has all been installed on a compacted 4" x 3" layout. TIM comes complete with:

* Step-by-step user installation guide
* Recommended RGB monitor guide, with detailed specifications and pinouts for many different monitors to ease in interfacing TIM to your RGB monitor.
* Graphic demonstrations displaying the power of TIM
* Our own GIF file viewer supporting the V9958
* Three disks packed with fairware software which support the V9938/58 chips.

Also included with TIM is the improved SOB (son of a board), a $50 value, and after reading over the in-depth review of the V9958 processor, we are sure you will find that TIM is the best thing ever for your TI! It costs $179 U.S. The SOB is an internal plug-in 2" x 2" PC Board with a 16K EPROM and two ASIC's which allows you to replace GROMs 0 and 1 in your TI console with OPA's own operating system featuring:

* 100% compatibility with the 99/4A operating system
* Updated to be compatible with V9938/58 devices
* True lower case to be added to all modules and programs
* 4K micro-manager for cataloguing floppy, ramdisk, hard drives, and from the catalog; delete, view files and run Assembly, Object, Forth or C99 programs
* System on power-up

The installation was easy enough as the instructions were complete and detailed. All that needs to be done is unplug the 9918 chip and two grom chips, plug in the boards, solder one wire from each board to the TI board and smoke test. The hardest part was cutting an opening in the back of the case for the 25-pin DB plug. This plug is also the one thing that one has to be careful with. Even though it is securely mounted to the TIM board, the only one thing holding the TIM board in place is the pins going into the 9918 socket. This is unavoidable, but one cannot insert or remove the plug without holding the socket which can only be done with the main power board removed from the case, otherwise damage may occur to the pins that go into the 9918 socket. That is the bad part. There is no trouble using Basic, nor is there the massive incompatibility problem. Some programs that do not follow the proper rules will not work, though, which are probably the same programs that will not work on 9640... but this is explained in the instructions.

The design and construction of these cards is very professional. OPA does top-notch work! TIM, SOB fire up right away when the console is turned on. There is no waiting for them to get their bearings.

As for SOB, it contains three socketed EPROMS that can be updated if need be. It also contains a Micro-manager that lists all the drives and the complete menu from whatever cartridge is installed. It will list all the programs available from a gram device if it is active. One may catalog a drive and run a program from that drive, or run a program from a cartridge, or gram device. I have not had any problems with any Micro-manager functions. It also gives the prompts in a digitized voice. SOB contains a new character file (true lower case) and replaces the TI logo with the OPA logo.

Both of these cards work fine with the Horizon ramdisk and P-Gram + cards, and have the same operational priorities as the original TI system. Both TIM and SOB have 1 year warranties and it is no secret that Gary Bowser backs up his products. I am very happy with the purchase and it is good to have 80 columns. There are two SSSD disks with programs included with the purchase. o

continued from page 7

```
380 CALL KEY(0,K,S):: IF K<49 OR K>53 THEN 380 ELSE K=K-
48 :: IF K=5 THEN DISPLAY ERASE ALL :: STOP
390 DISPLAY AT(13,1):T$(K): :"Input File: DSK": :"Outp
ut File:  DSK": : : :
400 ACCEPT AT(15,18)BEEP:I$
410 ACCEPT AT(17,18)BEEP:W$
420 !
430 ! OPEN FILES & INIT
440 !
450 DISPLAY AT(19,1):"Working . . . ."
460 IF K>2 THEN OPEN #1:"DSK"&I$,INPUT ELSE OPEN #1:"DSK
"&I$,INPUT ,FIXED 128
470 IF K=4 THEN OPEN #2:"DSK"&W$,OUTPUT,FIXED 128 ELSE O
PEN #2:"DSK"&W$,OUTPUT
480 A=1 :: W$="" :: ON K GOTO 720,570,490,650
490 !
500 ! DV80 -> DV80 ADD CR's
510 !
520 LINPUT #1:I$ :: GOSUB 210 :: IF EOF(1)THEN 550
530 IF A THEN IF P THEN PRINT #2:I$;C$ :: GOTO 520 ELSE
Q$=I$ :: A=0 :: GOTO 520
540 IF P THEN PRINT #2:Q$;C$:I$;C$ :: A=1 :: GOTO 520 EL
SE PRINT #2:Q$ :: Q$=I$ :: GOTO 520
550 IF A=0 THEN IF P THEN PRINT #2:Q$;C$ ELSE PRINT #2:
Q$
560 PRINT #2:I$;C$:C$ :: GOTO 250
570 !
580 ! DF128 -> DV80 NO CR'S
590 !
600 LINPUT #1:I$ :: W$=W$&I$:: K=1 :: S=LEN(W$)
610 IF SEG$(W$,K,1)=Z$ THEN 250 ELSE IF K>S THEN IF EOF(
1)THEN 250 ELSE W$="":: GOTO 600
620 P=POS(W$,N$,K):: IF P THEN PRINT #2:SEG$(W$,K,P-K)::
K=P+2 :: GOTO 610
630 P=POS(W$,Z$,K):: IF P THEN PRINT #2:SEG$(W$,K,P-K)::
GOTO 250
640 W$=SEG$(W$,K,255):: IF EOF(1)THEN PRINT #2:W$ :: GOT
O 250 ELSE 600
650 !
660 ! DV80 -> DF128
670 !
680 LINPUT #1:I$ :: IF ASC(I$)=128 THEN I$=" "
690 W$=W$&I$&N$ :: P=LEN(W$)
700 IF P>128 THEN PRINT #2:SEG$(W$,1,128):: W$=SEG$(W$,1
29,255)
710 IF EOF(1)THEN PRINT #2:W$&Z$ :: GOTO 250 ELSE 680
720 !
730 ! DF128 -> DV80 ADD CR'S
740 !
750 LINPUT #1:I$ :: W$=W$&I$:: K=1 :: S=LEN(W$)
760 IF SEG$(W$,K,1)=Z$ THEN 820 ELSE IF K>S THEN IF EOF(
1)THEN 820 ELSE W$="" :: GOTO 750
770 P=POS(W$,N$,K):: IF P THEN I$=SEG$(W$,K,P-K):: K=P+2
ELSE 800
780 GOSUB 210 :: IF A THEN IF P THEN PRINT #2:I$;C$ :: G
OTO 760 ELSE Q$=I$ :: A=0 :: GOTO 760
790 IF P THEN PRINT #2:Q$;C$:I$;C$ :: A=1 :: GOTO 760 EL
SE PRINT #2:Q$ :: Q$=I$ :: GOTO 760
800 P=POS(W$,Z$,K):: IF P THEN I$=SEG$(W$,K,P-K):: GOTO
820
810 W$=SEG$(W$,K,255):: IF EOF(1)THEN I$=W$ ELSE 750
820 IF A=0 THEN GOSUB 210:: IF P THEN PRINT #2:Q$;C$ EL
SE PRINT #2:Q$
830 PRINT #2:I$;C$:C$ :: GOTO 250
```

o

TIsHUG Shop with Percy Harrison

Once again we have been very fortunate in getting hold of some more Wang colour monitors and all back orders have now been filled. There are still eight monitors left in stock at the time of writing this article, so any member still wanting to add a monitor to their system should contact me as early as possible as they will be issued in the same order as I receive payment, which is $110.00 plus packaging and postage. Because of the physical size and weight of the monitor I would recommend that members within the Sydney metropolitan area wanting a monitor contact me to make arrangements to pick it up from my residence, so as to avoid possible transit damage and additional cost, as most of these units are without boxes. They are all in very good condition and are guaranteed to be in good working order. Remember, if you are going to install it in place of a TV set, you will need the interface card available from the shop. If it is to run on a system incorporating the TIM or Mechatronics 80 column cards, then the interface card is not required.

At the risk of shaming some of our members, I am pleased to report that I did receive a letter commenting on the quality of the software available from the shop and would like to point out that the letter came from our only member in France. My sincere thanks to you, Pierre, for taking the time and effort to let me know what your thoughts were, regarding our software and service. Also at the last meeting I did receive verbal comments from two of our members regarding software quality; many thanks to Peter and Tom.

Club Software Disks

A145	Scott Adams Adventures (DSSD)	$2.00
A212	G Programming Language (SSSD)	$2.00
A214	PLUS! V1.0 (SSSD)	$2.00
A245	Telco V2.03 (DSSD)	$2.00
A249	Animation Demo's (SSSD, 2 disks)	$4.00
A261	Assembly Language Games (SSSD)	$2.00
A354	Microdex 99 V4.2 (SSSD)	$2.00
A380	Super-Cataloger (SSSD)	$2.00
A382	Boot (40 column vers, SSSD)	$2.00
A386	Boot (80 column & hard disk vers, SSSD)	$2.00
A399	Nuclear 99'er (SSSD)	$2.00
A405	1000 Words (SSSD)	$2.00
A430	Configuring Funnelweb (SSSD)	$2.00
A436	Hotbug (SSSD)	$2.00
A437	Nasty and Seggregation (SSSD)	$2.00
A438	More Assembly Games (SSSD)	$2.00
A439	Multiplan Exercises (DSSD)	$2.00
A448	TI Print Shop (TIPS) V1.7 (SSSD)	$2.00
A448A	TIPS Graphics #1 (SSSD)	$2.00
A448B	Grips (TIPS Companion, SSSD)	$2.00
A450	Funnelweb 4.40 (DSDD)	$2.00
A450A	Funnelweb 4.40 (SSSD, 3 disks)	$4.00
A451	Multiplan V4.02 (SSSD)	$2.00
A453	The Nutcracker Suite (SSSD)	$2.00
A456	Rememberance Music (SSSD)	$2.00
A462	Rediskit (SSSD)	$2.00
A463	TI-EXAM (SSSD)	$2.00
A464	II Pastor Fido Vivaldi (DSSD)	$2.00
A465	Lute Music (SSSD)	$2.00
A466	Best of CONNI D.O.M. #54 (DSSD)	$2.00
A467	The Singing TI (SSSD)	$2.00
A468	Speech #1 (TEII, SSSD)	$2.00
A472	TI Writer Supplement (SSSD)	$2.00
A473	DM 1000 V5.0 (SSSD)	$2.00
A474	GIF Pictures (80 column card, DSDD)	$2.00
A481	ARTCON (DSSD)	$2.00
A481A	ARTCON (SSSD, 2 disks)	$3.00
A482	Horizon Utilities (SSSD)	$2.00
A483	TI Tiler (SSSD)	$2.00
A484	Mac-Labels V2.0 (SSSD)	$2.00
A485	YEO (SSSD)	$2.00
A486	Genealogy Record Keeping V1.12 (DSSD)	$2.00
A487	XHi (80 column card, DSDD)	$2.00
A488	LA 99'ers Utilities (SSSD)	$2.00
A489	FONTART #1 (TIA, SSSD)	$2.00

Tigercub Disks

TCC1	Tigercub Collection #1 (SSSD)	$2.00
TCC2	Tigercub Collection #2 (SSSD)	$2.00
TCC3	Tigercub Collection #3 (SSSD)	$2.00
TCC4	Tigercub Collection #4 (SSSD)	$2.00
TCC5	Tigercub Collection #5 (SSSD)	$2.00
TCC6	Tigercub Collection #6 (SSSD)	$2.00
TCC7	Tigercub Collection #7 (SSSD)	$2.00
TC 911	Display Calculator (SSSD)	$2.00
TC1015	Word Processing Utilities (SSSD)	$2.00
TC1122	Screen Fonts (Archived, DSDD)	$2.00
TC1131	Gemini Printer Utilities (SSSD)	$2.00
TC1145	Telecommunications Aids (SSSD)	$2.00
TC1210	Graphics Printing (SSSD)	$2.00
TC1211	TI Artist Pictures #1 (SSSD)	$2.00
TC1212	TI Artist Pictures #2 (SSSD)	$2.00
TC1213	TI Artist Pictures #3 (SSSD)	$2.00
TC1219	Ray Kazmer's Xmas Card (SSSD)	$2.00
TC1220-1229	TIPS Graphics (Archived, 10 DSSD)	$20.00

Refer to last month's magazine for availability of hardware and modules.

Packaging and postage charges:

	Surface	Airmail
1 to 2 Disks	$1.90	$1.90
3 to 9 Disks	$2.90	$3.60
10 - 20 Disks	$3.90	$4.80
TI Artist Plus	$3.00	$3.70
TI Base	$3.00	$3.70
TI Sort	$3.00	$3.70
5.25 inch half-height drive (1.25 kg)	Refer to your local post office	

Bye for now. o

Jenny's Younger Set

G' day gang! I have a surprise for you this month. The executive has approved a 'quiz competition'. The object is to see who can write the best program for a game. Anyone can enter, but I am more interested in receiving something from the 'younger set', let's say, under 18. It must be something original, not just a reprint of a game from a past issue of the magazine or something like that. You may want to look at a few games for ideas but the finished product must be your own. Please end your entry on a cassette, as no entry will be judged on a disk. The deadline is the 30th of September and since the idea for the competition is Vincent Maker's, we are going to let him do the judging. You can send it to him at the following address:

QUIZ COMPETITION
103 Rausch Street
TOONGABBIE, 2146

or give it to the Editor at one of the club meetings. You will probably need Extended Basic to give yourself the best chance and what follows is a 'mini' game given to me by Vincent to give you an idea of some commands that you will need:

```
100 RANDOMIZE
110 B=INT(RND*10)
120 INPUT A
130 IF A=B THEN PRINT "RIGHT" :: GOTO 150 :: ELSE IF A>B
THEN PRINT "TOO BIG" ELSE PRINT "TOO SMALL"
140 GOTO 120
150 END
```

Let's see a good turn out for this competition. We have not heard from a lot of you younger set for a while and I know you are around as I see some of you at the club meetings. We have an idea about the prize for the winner but will keep you posted on this in the coming issues. Be sure to be watching! JENNY o

TIsHUG Software Column

by Rolf Schreiber

Software Releases for June 1992

This month I am releasing a total of eight disks, including three from Jim Peterson's Tigercub Software. I hope that everyone will find something either useful, or to their liking, among this month's software. If you do not find any of these disks worth buying, then please let Percy Harrison know by phone or letter what your personal software needs are, and we will try to accommodate you, if at all possible.

DISK A420 contains 5 ATARI modules, converted to run out of Extended BASIC, on systems fitted with 32K memory expansion. The games include such favouites as Jungle Hunt, Donkey Kong, MS Pacman, Shamus and Picnic Paranoia. The games are selected from a menu and automatically load with a single keypress. A great disk for our newer members and those with young children.

A420 Diskname: GPL/12 Format: SSSD

Filename	Size	Type / Length	
DONK	33	PROGRAM	8192
DONL	18	PROGRAM	4108
JUNG	2	PROGRAM	155
JUNH	33	PROGRAM	8192
JUNI	33	PROGRAM	8192
LOAD	9	PROGRAM	2037
MPM1	33	PROGRAM	8192
MPM2	33	PROGRAM	8192
PICNIC1	33	PROGRAM	8192
PICNIC2	33	PROGRAM	8192
SHAM	33	PROGRAM	8192
SHAN	15	PROGRAM	3584

DISK A431 is called Object Linker V3.0 and was written by Art Green, the same person who produced the major upgrades for TI Writer and Multiplan. This disk is allows you to convert object files (DIS/FIX 80 format) into program files (memory image format). Whilst being mainly for programmers, the disk contains extensive documentation which should be enough to guide most people through the conversion process.

A431 Diskname: LINKERV3 Format: SSSD

Filename	Size	Type / Length	
LINKER	33	PROGRAM	7952
LINKES	8	PROGRAM	1700
LNKDOC	8	DIS/VAR	80
LNKDOC1	63	DIS/VAR	80
LNKDOC2	73	DIS/VAR	80
LNKDOC3	43	DIS/VAR	80
LNKINST	16	PROGRAM	3826
LOAD	3	PROGRAM	504
LOADINST	3	PROGRAM	504
RAGLIB	14	DIS/FIX	80
README	3	DIS/VAR	80

DISK A475 is from the Hamilton User Group from Canada, and was compiled by Stephen Johnson. This disk is called Clubline-99 (V4 N8) and contains a hangman game (including a speech version), a geography quiz on Canada, a lucky numbers selector, a graphics demo, a chapter on learning assembly language, and a very useful disk sector reading utility (with source code).

A475 Diskname: VOL4/8 Format: SSSD

Filename	Size	Type / Length	
BILLPAYER	13	PROGRAM	3048
CANADA	53	INT/VAR	254
DISKREAD	5	PROGRAM	786
DISKREADS	30	DIS/VAR	80
DRAWHORP	7	PROGRAM	1416
DRAWHORSP	45	DIS/VAR	80
HANG/FILE	36	PROGRAM	8864
HANG/WORD1	6	INT/FIX	192
HANG/WORD2	6	INT/FIX	192
HANG/WORD3	8	INT/FIX	192
HANG/WORD4	8	INT/FIX	192
HANGMAN	36	PROGRAM	8857
HANGMANSP	36	PROGRAM	8879
LOAD	13	PROGRAM	3022
MATHP	4	PROGRAM	726
PLAN/MAKER	9	PROGRAM	1991
YLOAD	10	DIS/FIX	80

DISK A90 is the companion disk to A489 and contains the second and third volumes of TI-Artist fonts from Paul Scheidemantle. This disk is DSSD and needs a version of TI-Artist to display the fonts.

A490 Diskname: FONTS2&3 Format: DSSD

Filename	Size	Type / Length	
--README--	4	DIS/VAR	80
-README-	4	DIS/VAR	80
3DSM_F	26	DIS/VAR	80
BIGBLOCK_F	34	DIS/VAR	80
BLACK_F	33	DIS/VAR	80
BLOCK_F	34	DIS/VAR	80
BOLD_F	27	DIS/VAR	80
BROADWAY_F	33	DIS/VAR	80
COMP_F	22	DIS/VAR	80
FAROUK_F	39	DIS/VAR	80
HEADLNR1_F	40	DIS/VAR	80
HEADLNR2_F	13	DIS/VAR	80
LCHR/L_F	35	DIS/VAR	80
OLDE/SL_F	27	DIS/VAR	80
OLDE/SU_F	42	DIS/VAR	80
SAMPLER1_F	36	DIS/VAR	80
SAMPLER2_F	11	DIS/VAR	80

DISK A91 is part of the Boston Computer Society's disk library and was written by J. Peter Hoddie, who is well known as a talented programmer in the TI99/4A community. This program allows you to design graphics screens in the Extended BASIC environment, and the disk comes with comprehensive instructions.

A491 Diskname: BCS11 Format: SSSD

Filename	Size	Type / Length	
CHAR	45	PROGRAM	11161
LOAD	3	PROGRAM	512
READ-THIS	22	DIS/VAR	80
SAMPLE	18	INT/FIX	128
WRITER	8	PROGRAM	1632

Tigercub Software Releases

TCC-7 is the seventh disk in Jim Peterson's Tigercub Collection. This disk contains 14 games which can all be selected from the menu shown below:

1 CAROM - shoot a billiard ball into the pockets
2 PLANETARY DEFENSE - space shooting game
3 GETAWAY - avoid getting caught
4 SHEEP DOG - round up the sheep
5 WHITE KNIGHT - on a quest to collect treasures
6 BARS AND BALLS - drop balls; avoid obstacles
7 AHAMUR - govern the city of Sumeria
8 ARTILLERY BATTLE - shooting game
9 BOWLING - nine pin bowling; up to 20 players
10 ROMEO AND JULIET - rescue Juliet
11 SKY RESCUE - rescue captives from terrorists
12 TI-TROGMAN - a maze game in TI BASIC
13 VERSAILLES - a maze game in TI BASIC
14 ZAN QUEST - a game of conquest

TC-911 is a new program from Jim Peterson which emulates a very sophisticated 6-window, 6-memory, 34-function, 14-digit display calculator on your TI99/4A. Two versions of the program are included on the disk, a

continued on page 20

TI-Bits Number 16

by Jim Swedlow, CA USA

[This article originally appeared in the User Group of Orange County, California ROM]

HOW YOUR TI SAVES TEXT FILES

In our TI world, most text is saved in Display Variable 80 (DV80) files. This is what the TI Writer Editor uses. What is DV80?

Display means that the file is saved using ASCII characters. If you use a disk sector editor to look at a DV80 file, you will see that the text looks just about the same as it was written. Internal files are not always as easy to read (but that is another column).

A file is made up of records. In a DV80 file, each record contains one line of text as it appears in the Editor. Variable means that each record is only as long as the text line.

Consider these two lines of text:

```
TI
99/4A
```

When this is saved on disk, there will be two records in the file. The first record is "TI" and the second is "99/4A". Each record is preceded by the number of characters in the record in Hex. Hex FF is used to mark the end of the file. The file would look like this:

```
Hex  02 54 49 05 39 39 2F 34 41 FF
ASC     T  I    9  9  /  4  A
```

In a Display Fixed 80 (DF80) file, each record still contains one text line but is exactly 80 characters long. Your 4A pads each record by adding the required number of spaces to the end of each text line.

WHY YOU NEED TO KNOW
HOW THE REST OF THE WORLD SAVES TEXT FILES

As a loyal TI user you may not think that you need to know how others (MS-DOS, CPM, etc) save text files. If you do any work with modems, however, you do.

The reason is that you may down load a text file and find that it is Display Fixed 128 (DF128). Why? There is a standard protocol in the TI world for transferring files using XMODEM. It was designed by Paul Charlton (creator of FAST TERM). The first record is NOT the first line of text. Instead, it is the disk header sector (which describes the file in a manner than can be read by the disk controller).

If the first record is not the header, however, your modem program (TELCO, FAST TERM, MASS TRANSFER, etc) assumes that you are talking to a non TI system and will save the file as DF128.

The reason, then, that you need to know how other systems save text files it that you may get one.

HOW THE REST OF THE WORLD SAVES TEXT FILES

The short answer is DF128. But there is more. Unlike a DF80 file, there is no padding. Instead, all of the text lines are run together. The end of each text line is marked with a carriage return <CR or CHR$(13)> and a line feed <LF or CHR$(10)>.

One record may have one, two or more text lines, each ending with a CR and LF. If there is not enough room left in a record for the next text line, enough of the line is added to the record to bring it to 128 characters. The rest of the text line starts the next record - followed by a CR and LF. The end of the file is marked with CHR$(26), which in the IBM and CPM worlds is <CTRL Z>.

Remember our sample text?

```
TI
99/4A
```

Since it well under 128 characters, the file will only contain one record:

```
Hex  54 49 0D 0A 39 39 2F 34 41 0D 0A 1A
ASC   T  I       9  9  /  4  A
```

Hex 0D 0A is a CR and LF. Hex 1A is the end of file marker CHR$(26).

CONVERTING FILES

There are a number of programs that convert files from DF128 to DV80 or from DV80 to DF128. Some of the assembly ones are quite fast. There should be a program in this issue called CONVERT. It does those conversions and two others.

A little background. Sometimes you may look at a file and notice that there are no CR's. If you reformat such a document, everything will be jumbled into one big paragraph. TI Writer stops reformatting when it hits a CR. FUNNELWEB stops when it hits a CR or a blank line. Either way, the document is a mess.

CONVERT, when converting a DF128 file to DV80, can add a CR to blank lines, to the end of paragraphs and to lines that start with a period (Formatter commands). This takes a little longer but it makes the file much easier to edit. Also, CONVERT can add CR's to DV80 files that lack them.

NOTE TO OTHER USER GROUPS

I have now written 20 XB Columns and 20 in the TI BITS series. From time to time, other user groups have published some of my work in their news letters. If anyone wants a complete set, please send me two (2) DSSD disks or four (4) SSSD disks, a return mailer and return postage. My address is 7301 Kirby Way, Stanton, CA 90680.

Enjoy.

```
100 ! CONVERT
110 ! Version 1.0
120 ! 09 Aug 88
130 ! By Jim Swedlow
140 ! Based on XPREP by Carl Walters
150 !
160 DISPLAY ERASE ALL:: CALL SCREEN(5):: FOR A=0 TO 14
:: CALL COLOR(A,16,1):: NEXT A
170 FOR A=1 TO 4:: READ T$(A):: NEXT A
180 N$=CHR$(13)&CHR$(10):: Z$=CHR$(26):: C$=CHR$(13):: G
OTO 300
190 DATA DF128 -> DV80 add CR's,DF128 -> DV80 no CR's,
DV80 -> DV80 add CR's,DV80 -> DF128
200 CALL KEY :: Q$,S,P,K,I$,W$ :: !@P-
210 !
220 ! STRING CHECK SUB
230 !
240 P=1 :: IF I$=" " OR I$="" THEN I$="" :: RETURN ELSE
IF ASC(I$)=46 THEN RETURN ELSE P=0 :: RETURN
250 !
260 ! CLOSE FILES AND END
270 !
280 CLOSE #1:: CLOSE #2:: DISPLAY AT(19,1)BEEP:"DONE"
290 FOR P=1 TO 100 :: NEXT P
300 !
310 ! TITLE SCREEN
320 !
330 DISPLAY AT(5,5):"CONVERT Version 1.0": : : : : : :
:"Press For"
340 FOR S=1 TO 4 :: DISPLAY AT(14+S,1):STR$(S);" ";T$(S)
:: NEXT S :: DISPLAY AT(19,1)BEEP:"5 End Program"
350 !
360 ! PICK FUNCTION
370 !
```

continued on page 4

XB tips Number 17

by Jim Swedlow, CA USA

[This article originally appeared in the User Group of Orange County, California ROM]

SPEEDING UP EXTENDED BASIC

I have been looking for ways to speed up Extended Basic programs and have found some interesting things.

Most of the information in this article is based on a FOR NEXT loop. To compare, for example, <PRINT A> with <PRINT A;>, I ran a program like this:

```
10 INPUT A$
20 FOR A=1 TO 1000
30 PRINT A
40 NEXT A
```

Line 10 is <INPUT A$> to make sure that the pre-scan time did not skew the results and to give me a marker to start my stopwatch.

I ran the program three times and then averaged the run times. Next I changed line 30 to:

```
30 PRINT A;
```

Again, I ran the program three times and averaged the run times. Then I did the same thing with DISPLAY AT. The results were:

Line 30	Run Time
PRINT A	105.5 seconds
PRINT A;	57.6 seconds
DISPLAY AT(5,5):A	66.3 Seconds

The loop with no line 30 took about 10.6 seconds. Therefore, PRINT A takes a little less than a tenth of a second to execute.

What follows are some things that I tried and the results. Some conclusions challenge advice I have read and some is new.

REMarks AND ! TAILS

Most of the material I have read suggests that removing REM/! lines and ! tails will improve execution time. It will, but not by very much.

A plain 1 TO 1000 loop took 10.6 seconds. Adding either a REM or ! line INSIDE the loop increased execution time to 11.2 seconds. The same applied to a ! tail.

Removing these will only slightly speed up your program.

INSIDE ARRAYS

I wondered if it took longer to access one member of an array versus another. I DIMentioned an array at 200 and then looked at different members.

I expected to find some relationship between the number and the time (faster to access the beginning or end). What I found was that the run time depended on the size of the number inside the parenthesis:

Array Members	Run Time
B(1) to B(9)	15.3 seconds
B(10) to B(99)	15.6 seconds
B(100) to B(200)	16.0 seconds

Apparently the more digits a number has, the longer it takes Extended Basic to read and digest it. Further, it takes about the same amount of time to access any member of an array.

DEFinitions ARE GLACIAL

I knew from experience that DEFinitions were time consuming but I was surprised to find out just how slow they are. I ran a loop with a calculation done through a DEF and then without the DEF. The DEF loop was 55 seconds slower than the non-DEF loop.

It takes a long time (in computer terms) for your 4A to find a DEF (NOT counting the actual time to execute the calculation). Avoid DEFinitions!

ORDER OF VARIABLES

One of the things your TI does during pre-scan is to build a variable table. This is a table of all of the variables in the program and the memory location of each variable's current value.

Each time your program uses a variable, it first searches the table for the variable and then goes to the memory location to find the value.

Our TI's list variables in reverse order. The first variable it finds in a program is at the end of the table and the last variable found is at the beginning.

I created a program with 26 variables and then used one of them inside the loop. Times were:

Variable used	Run Time
First in program	19.0 seconds
Last in program	12.4 seconds

Your TI finds your variables in either the order of use or whatever order you placed them in the pre-scan list. These results suggest that if you change the order so that infrequently used variables appear first and those that are used often appear last, your execution time should decrease.

To test this, I took a program that had the variables listed in alphabetical order. I rearranged them to reverse order of use. The results were impressive:

Variable Order	Run Time
Alphabetical	14.3 minutes
Reverse use	12.5 minutes

This step reduced run time 1.8 minutes or 12% percent. NOT BAD!

LINE NUMBERS

Your TI also uses a line number table. This table is similarly in reverse order (first line last, last line first, etc). When you use a line number in your program (GOTO, etc), your 4A must search the line number table to find the memory location of the line contents. Then it reads the line instructions, crunches them and executes.

I constructed a program that looks something like this:

```
10 INPUT A$
20 FOR I=1 TO 1000
30 ??????
40 NEXT I
50 RETURN
-----------------------
Lines 60 through 2050
   - 200 lines -
  are all REM lines
<60 REM>, <70 REM>, etc
-----------------------
2060 RETURN
2070 SUB A :: SUBEND
```

continued on page 18

To See or Not to C

by Geoff Trott

This is the second article in a series on the language C and its implementation on the TI99/4A by Clint Pulley. In the first article, I gave a simple program and told you how to compile, assemble and load it in the Funnelweb environment. The program and the script loader file for using it will be repeated here for easy reference.

```
AUTO
ASSM
FILE "DSK1.FIRSTC;O"
STOP
FILE "DSK1.CSUP"
FILE "DSK1.CFIO"
FILE "DSK1.PRINTF"
LAST START
```

```
/* A first program in c99 by Geoff Trott */
/*                                       */
#include "DSK1.STDIO"
extern printf();
main()
{
  printf("Hello Geoff");
  exit(0);
}
```

The first thing to note is that the C language is relatively unstructured. That is, lines do not mean very much. Statements can be on more than one line (we are used to that in BASIC with a 28 column screen) and more than one statement can be on the same line (as in Extended BASIC using the statement terminator ::). All this means that statements must have a terminator, which in the case of C is ";". Groups of statements for procedures, loops or conditionals are placed within curly brackets "{ }" to indicate the start and end of the group of statements. So in the simple program above, the two executable statements, "printf" and "exit" are within the "{ }". This is identified as the main procedure by the main() statement preceding the brackets. All the rest of the indenting and spacing is in the interests of readability. The whole procedure could be on one line like:

```
main(){printf("Hello Geoff");exit(0);}
```

The first two lines in the program are comments. A comment starts with the two character sequence "/*" and ends with the two character sequence "*/". Like other C statements, a comment can go over one line or be on a line with other statements. Anything can be in a comment other than another comment. You cannot nest comments. The third line tells the compiler to read in the file specified. This will be some C code or statements and is why the compiler reports more lines processed than are in the program you typed in. The fourth line tells the compiler to tell the assembler that this function will be loaded in later with the rest of the support functions, in this case in "PRINTF". These routines are in what are normally called "libraries" of compiled (and assembled) code.

Note that most words are in lower case. The case of letters is significant in C. The usual thing is to use lower case mostly, with occasional use of upper case. However, as the output of c99 goes to the assembler which only recognises upper case letters except in comments and text fields, names of procedures must be eventually converted to upper case for the assembler. This means that names which only differ by the use of the case of the letters (TesT and test say) will end up being the same. Also procedure names must differ in the first 6 characters, also for the assembler, although they can be much longer than that for C itself. In fact the compiler only looks at the first 6 characters of any name.

So how do you learn to use a new language? One of the good ways is to look at some program that someone else has written and try to see if you can work out what has been done. In the case of C, this can be done by looking at the examples on the release disk where there are both source files and some documentation to go with some of them. As an example, I shall take the string routines as I have spent some time getting them to work properly. There are 4 files associated with these procedures. STRINGDOC contains the documentation of the procedures, what they do and how to use them. STRING;C is the source code of the procedures, STRINGS contains this code compiled and assembled while STRINGI is an include file which contains the definitions needed for the procedures to be used.

A look at the first three procedures in the documentation file gives the following information.

Length of string:

```
strlen(s)
char *s;
```

Compare two strings:

```
strcmp(s1, s2)
char *s1, *s2;
```

This returns an integer which is less than zero if s1 is "less than" s2; zero if s1 is identical to s2; and greater than zero if s1 is "greater than" s2.

Compare n characters of two strings:

```
strncmp(s1, s2, n)
char *s1, *s2;
int n;
```

This returns an integer which is less than zero if the first n characters of s1 are "less than" the first n characters of s2; zero if the first n characters of s1 are identical to the first n characters of s2; and greater than zero if the first n characters of s1 are "greater than" the first n characters of s2.

The above paragraphs briefly describe three procedures which all return a value based on the length of a string for the first one and a comparison between two strings in the case of the last two. I guess one of the first points of discussion should be about strings, what are they?

Strings are arrays of characters or vectors of characters or groups of characters or words or sentences. They are a number of characters which are grouped together and stored in sequential memory locations and referred to by one name. In C, as in all modern languages, all variables must be declared to have a particular type before they are used. For strings, the variable type is "char" and the string must be declared as an array of sufficient size to be able to hold all the characters plus one more. This extra character is used to hold an "end of string" character. In BASIC, the strings have an extra character which contains the count of the number of characters in the string and is the first character stored in the string storage area. In C, the "end of string" character is the last character stored in the string storage area.

Let us now have a look at the code for these three procedures. This is in the file STRING;C and the start of it follows.

```
/*
** string function library
**
** contributed by :
** Tom Wible
** 203 Cardinal Glen Circle
** Sterling, VA
** USA 22170
**
```

```
** Modified by Geoff Trott, 29 June 1989
** 20 Robsons Road, Keiraville, NSW 2500, Australia
**
*/
#define NULL 0
#asm
  DEF STRLEN,STRCMP,STNCMP,INDEX,RINDEX
  DEF STRCPY,STRCAT,STNCPY,STNCAT
#endasm
strlen(s) /* returns string length*/
char *s;
{
  int n;
  n = 0;
  while (*s++) n++;
  return (n);
}
```

The first few lines are comments and so can be skipped over. Then a define statement sets the name NULL to be zero. This is like an EQU statement in assembler, except can be used more generally than in the assembler. NULL will be the "end of string" character for the strings. #asm ... #endasm are statements which surround assembler code. Whatever is between these two words is sent straight to the assembler without the compiler doing anything to it. In this case the names of all the procedures in the file are (in upper case) are DEFined ready for the loading process. This is followed by the first function, strlen(s), to return the length of the string. Note the use of comments on the end of the line with the function name.

This function counts the number of characters in the string s. This is done by the "while" statement, using "n" as a counter. The value of "n" is returned as the length of the string. Now let us examine some of the details. The code of the procedure is between the "{ }" and starts with the declaration that "n" is an integer. This is one of the two base data types for c99, along with "char" for characters. Then "n" is initialised to zero and the "while" loop is entered. A "while" loop executes until the logical expression in parentheses becomes false. The while loop consists of one statement, "n++". If there were more statements that were to be executed in the "while" loop, these would be enclosed in "{ }". The "n++" statement uses one of the useful C operators, increment. Instead of having to say "n=n+1", "n++" does the same thing. It is a bit more useful in other contexts also. So the "while" loop is simply incrementing the counter "n" until the expression "*s++" is false. This is where the explanation becomes a bit more difficult.

Return back to the start of the procedure "strlen(s)". What exactly is "s"? Well we would expect it to be the string, except that a string is an array of characters and we can only pass a single entity to a procedure. What is actually passed is the address of the first character in the string. This is called a pointer in C. A pointer is a variable which contains an address which points to the data. A pointer is most useful for arrays but can be used for any data. A pointer has to be identified with the type of the data which it is addressing so that when it is incremented it will point to the next item in the data. This is like the auto-increment in registers where if a byte is moved the register contents are increased by one but if a word is moved the contents are incremented by two. This is done in this case by declaring that the data pointed to by the pointer is a character. "*s" returns the character pointed to by "s". The statement "char *s;" declares that the pointer s is a character pointer.

Following on from that, the "while" conditional clause of (*s++) returns the character pointed to by "s" which will have a non-zero ASCII code (and so be treated as TRUE) until the last character in the string appears which is the "end of string" character and is defined to have an ASCII code of zero which is also the value of FALSE. The "++" operator increments the pointer each time it is executed and because it appears after the variable, it is done after the character is returned. The same statement could be written as "while(*s++ != NULL)" which is translated as while the character is not equal to NULL, increment the pointer and do the following statement. This then covers all the statements for this function. It simply counts all the characters in the string until it reaches the "end of string" character and returns that value. Notice that the parameter that is passed in the procedure call is declared outside the "{ }" that enclose the procedure code while the variables used in the procedure are declared inside the "{ }" and are local to the procedure.

```
strcmp(s1, s2) /* compares s1 to s2, returns n<0
                  if s1 before s2, etc */
char *s1, *s2;
{
  int r12;
  for ( ; (*s1) != NULL; s1++, s2++) {
    r12 = (*s1) - (*s2);
    if (r12) return (r12);
  }
  r12 = (*s1) - (*s2);
  return (r12);
}
```

The next procedure compares two strings and returns zero if they are identical and a non-zero value if they are not. In this case pointers to two strings are passed to the procedure and these are declared as pointers to "char"s. Inside the procedure an integer variable is needed so it is declared first. The work is done with a "for" loop with two statements in the "for" loop enclsed in "{ }". The indenting is to make it more obvious where the loop is. In this case the difference between the ASCII codes of the characters is calculated and if this is not zero a return is made with that difference as the returned value. So as soon as the strings differ the procedure ends and a non-zero value is returned. If the strings are the same, there must be a way of identifying when to finish the loop. This is in the conditional clause of the "for" function. Inside the parentheses following the "for" statement there are three groups of statements separated by ";". The first group of statements are executed once at the start of the loop and used to initialise variables. This is like the first number in the BASIC statement FOR I=1 TO 10 STEP 1. First I is initialised to 1. In this case there is nothing to be initialised so this is blank but the ";" is still required. The second set of statements define the terminating condition of the loop (like the second number in BASIC). In this case it is when the "end of string" character appears in the first string. The third group of statements are executed once at the end of every time through the loop and are usually used to define how the variables are to be changed (like the STEP statement). In this case the pointers to both strings are incremented to point to the next characters in the strings. So the "for" loop causes the characters in both strings to be compared until either there is a difference or the first string finishes.

The final statements cover the case that the "for" loop finishes normally. In this case either both strings are identical and the difference in the two character codes will be zero (both NULL) or the second string will not be finished so that it will be longer than the first and so greater and the result returned will be negative.

```
stncmp(s1, s2, n) /* compares n chars of s1 to s2,
                     returns n<0 if s1 before s2 */
char *s1, *s2;
int n;
{
  int r12,i;
  if (n < 1) return(0);
  for ( ; n > 1; --n, s1++, s2++) {
    r12 = (*s1) - (*s2);
    if (r12) return (r12);
    if (*s1 == NULL) return (r12);
  }
  r12 = (*s1) - (*s2);
  return (r12);
}
```

continued on page 17

Writing in Machine Code

by J.E. Banfield

For the serious machine code programmer, you will need to have available:

1. A MiniMemory module or its equivalent
2. A copy of the TMS9900 Microprocessor Data Manual published by Texas Instruments Incorporated 1982

Also highly desirable:

3. TMS9918A/TMS9928A/TMS9929A Video Display Processor manual, TI 1982
4. For convenience, a calculator with hexadecimal functions, for example Casio College fx-100B.

For design and for writing interface programs you might need the TMS9901 and TMS9902 Data manuals.

==========

However, do not despair; this series of articles can be followed without the above. If necessary, machine code can be examined and entered from Extended BASIC using PEEK and POKE etc. I would not recommend it.

==========

All numbers in this series are in hexadecimal (or binary) unless otherwise stated or preceded by R.

==========

For convenience I keep a summary of machine operation codes in descending numerical order. This enables de-compilation of a program. For personal reasons, I use the DEC Macro 10 mnemonics or near equivalent rather than the TI Assembler mnemonics. This may irritate assembler programmers but has the advantage that you can clearly distinguish machine code descriptions from assembler, even though the latter as machine code ultimately.

Figure 1 gives my summary of the machine code with mnemonics, headed MacroT De-compiler. For each set of codes an address description (mode) is appended: D,S, d,S, c,s etc. I will explain as we go.

Each machine code instruction is organized as a two byte, 16 bit one or two 16 bit data words containing address information.

MiniMemory Module

For detailed instructions in the use of EasyBug, refer to the MiniMemory Command Module Data Book, pages 64 to 71.

==========

Plug in your MiniMemory module, turn on the power to your console and select EasyBug using Key 2. After you bring up "?" on the screen, type in M02B2 and then press enter. A line:

M02B2 = C8

comes up. Press the space bar to display the next line:

M02B3 = 0B

which is the next byte in memory. I have selected an address in the console 0000 to 1FFE segment (ROM); in fact the machine code entry for KSCAN (the GROM entry is at address 02AE). The instruction word is thus C8 0B.

Find C in Figure 1: "MOVE D,S"

This translates as: move the word at address defined by S to the location defined by D. (Note that in assembler such a move uses the operands in opposite order, a very good reason for using different mnemonics.)

The full bit pattern for the instruction is:

```
1100  1000  0000  1011
 C     8     0     B
```

Opcodes F to 4 leave 12 bits remaining after t opcode to specify two full 6 bit address specifications, first the destination address "D" followed by the source address "S". Thus the source address specification, the right hand six bits are:

00 1011 that is 0B.

The first two bits "0", specify the address mode as Ac; Accumulator (register in assembler). Thus the source address is Ac"B" (or R11), from which the data is taken by the processor for transfer to the destination address in memory specified by "D". The accumulators (registers) are in fact ordinary memory locations and can be used or addressed as in any other memory access. How their location is defined (for example by instruction 040) is left for another occasion. Once defined, these specified addresses can be addressed simply (4 bits) and can be used (except Ac"0") as index registers.

For this instruction the 6 bits specifying the destination "D" are:

1 0 0 0 0 0

The first two bits, 1 0 or 2, refer to the @.+2 mode. The second four bits, zero, indicate no index registration and so the destination address is to found in the next two lines:

M02B4 = 83
M02B5 = D8

so that the destination address is at M83D8 in the console RAM.

In summary: MOVE 83D8,B

The accumulator "B" (R11) has a special function in the TMS9900 processor in the it holds the return address for calling routines. Since the next instruction calls another subroutine it will Ac"B" for this purpose and so the original return address, now at M83D8 is saved for later use.

Next Instruction

Press the space bar twice more. The next instruction is thus revealed.

M02B6 = 06
M02B7 = A0 ; 06 A0

constituting the next instruction.

Look at Figure 1. The code 06A lies below the opcode 06C and above 068 and it "belongs to the 068 code. The bit pattern:

```
0000  0110  1010  0000
 0     6     A     0
\ 068 part  /\ S address part /
```

Once again the S address, of 2 type specifies @.+2 mode. The instruction code 068 or JSP is a jump and save program counter. The current contents of the program counter are transferred (saved in Ac"B" (R11)) and the contents of the source address are transferred to the program counter of the CPU. These contents are revealed by again pressing the space bar twice:

M02B8 = 08
M02B9 = 64

so that the JUMP address is 0864. We will not inquire at this stage what the subroutine at 0864 does; eventually it will return to the address saved to Ac"B" by the calling program and this will be M02BA. (Very much later, at the end of the KSCAN routine, the return address will be taken from M83D8.)

Thus, back from the 0864 subroutine, the processor reads the next instruction, space bar twice:

M02BA = 04
M02BB = CC

The instruction, from Figure 1, is coded 04C and is SETZS. This sets the source memory to zero (the contents of the source address are, in fact, ignored). The instruction:

```
0000  0100  1100  1100
\     04C   /\ 0C  /
```

splits up into a 10 bit opcode and a six bit source

continued on page 14

Scott Foresman Mathematics Series

Reviewed by Charles Good, OH USA

I cannot say enough nice things about this software. It is, in my opinion, far superior to the MILLIKEN MATH SEQUENCES TI cartridge software many of us are familiar with. When it comes to teaching students NEW mathematical concepts, as opposed to just reinforcing previously learned concepts with drills, there is nothing better than the SCOTT FORESMAN MATHEMATICS COURSEWORK SERIES. None of the basic Maths education software for Apple or MS-DOS computers is as good, in my opinion. You can actually sit a student down in front of the computer, start up the module, and have the computer do the complete job of teaching the student an unfamiliar Maths concept without any further human intervention. I have done so with my 1st grade daughter and 5th grade son, and I have talked to a couple of primary school educators who use 99/4As and who have confirmed that these cartridges are actually self teaching. I find this amazing! This software is computer assisted learning at its very best.

All cartridges in the MATHEMATICS COURSEWORK SERIES were written by Thomas Hartsig. They seem to be designed for in classroom use, but also can be used at home. Each makes liberal use of sprites and colour graphics, music, and especially speech, and each is based upon a particular theme. The powerup cartridge menu gives a choice of several activities, the last being a random review of the others. At the beginning of each activity, the student is given the choice of 1- A TEACHING EXAMPLE, or 2- PRACTICE EXERCISES. Selecting 1- really shows the magic of the cartridge. The problem is set up graphically on screen step by step. Digits float around the screen as the problem is solved, and while all this is happening the computer TALKS the student verbally and with on screen words through each step of the solution. You really have to see all of this to appreciate how good these tutorials are.

When 2- PRACTICE EXERCISES is selected a problem is displayed and the student is asked to solve it digit by digit, usually from right to left, just as the problem would be done with pencil and paper. A wrong answer gets a "try again" the first time. If the student waits too long the computer prompts on screen and verbally "your turn" a couple of times and if no solution is attempted the computer then solves the problem. If too many problems in a series are solved incorrectly or not at all the computer says "you can do better than this" and immediately begins to display some additional TEACHING EXAMPLEs.

The better known software in this series, cartridges that are not rare, include ADDITION AND SUBTRACTION 1, as well as ADDITION AND SUBTRACTION 2, MULTIPLICATION 1, and DIVISION 1. The activities of the "rare" c1983 cartridges are listed below. Both NUMERATION cartridges are listed in TI's last 1983 price list. None of the others are mentioned in official TI promotional literature.

NUMERATION 2- at the carnival.

This was available from some dealers in late 1983 but was less commonly available than Numeration 1.

TI, in its 1983 pamphlet "Texas Instruments Home Computer Program Library" lists this cartridge as appropriate for "late primary, 10-12 years".

1- 4 DIGIT NUMBERS. (Write four thousand two hundred five. Which digit is in the hundred's place?)
2- COMPARE NUMBERS. (Which of two is larger?)
3- Rounding numbers (To the nearest 10s with 5 or more rounding to the next digit.)
4- 5 AND 6 DIGITS
5- 7, 8, AND 9 DIGITS
6- DAILY USE OF NUMBERS (Weigh produce on a scale and round to the nearest 10 ounces.)
7- REVIEW.

MULTIPLICATION 2- Mighty Multiplication.

This has a cute graphic of a little flying super hero, Mighty, who saves the day if you answer correctly. I wrote about this module previously under the topic of "never released software" but I subsequently found it listed on some TRITON catalogs.

1- MULTIPLYING 10 AND 100. (3x10=? 3x100=?)
2- MULTIPLES OF 10 AND 100 (3x7=? 3x70=? 3x700=?)
3- 2 AND 3 DIGITS TIMES 1 DIGIT.
4- 2 DIGITS WITH RENAMING. (I used to call this "carrying".)
5- 3 DIGITS WITH RENAMING.
6- MORE THAN 1 RENAMING.
7- WORK PROBLEMS.
8- REVIEWING IT ALL

ADDITION AND SUBTRACTION 3

1- ADD 2 DIGIT NUMBERS.
2- SUBTRACT 2 DIGIT NUMBERS
3- REGROUP OBJECTS ("13 ones equals 1 ten and 3 ones.")
4- ADD WITH RENAMING
5- SUBTRACT WITH RENAMING
6- ADD 3 DIGIT NUMBERS
7- SUBTRACT 3 DIGIT NUMBERS
8- REVIEW.

NUMERATION 1- Under the Big Top.

TI considers this suitable for "early primary 5-7 years" The cartridge was available from some dealers by the end of 1983 but I never personally saw one in the stores.

1- NUMBERS TO 9
2- COMPARE NUMBERS (3<5)
3- HOW MANY TENS
4- NUMBERS TO 99
5- NUMBERS IN ORSER ("5 is one less than 6.")
6- ORDINAL NUMBERS (First, second, third, etc.)
7- NUMBERS TO 999
8- REVIEW

FANTASTIC FRACTIONS 1

1- WHAT IS A FRACTION
2- A FRACTION OF MANY ("How many boxes are black? How many total boxes are there? What fraction are black?)
3- EQUAL FRACTIONS (1/2 = 2/4)
4- MIXED NUMBERS (Whole number plus fraction, such as 2 1/4)
5- APPLICATIONS (Show 5 1/3 on ruler scale.)
6- REVIEW

DECIMAL DELI 2

There is reference to a DECIMAL 1 in the literature, but I have never seen DECIMAL 1.

1- PLACE VALUE (ones, tens, tenths, hundredths, thousandths; 92.475 equals "ninety two and four hundred seventy five thousandths")
2- COMPARING, ORDERING ("5.374 is greater than 5.334")
3- COUNTING PLACES (35.8284, four decimal places)
4- MULTIPLYING DECIMALS
5- ZEROS IN THE PRODUCT (0.04 x 0.1 = 0.004; "We need two zeros in front of the four to make three decimal places.")
6- APPLYING DECIMALS (Each roast beef sandwich costs $3.93. What is the cost of 7 sandwiches?)
7- REVIEW

o

Renew Now!

Programming Music part 2

by Jim Peterson, Tigercub Software, USA

In Part 1 I showed you how to set up a musical scale to create notes, and how to merge in various little routines to create a variety of musical effects, but I did not tell you how to figure out what numbers to put in between those GOSUBs. So, here is the little program that makes it all easy.

```
100 CALL CHAR(127,"000F080F0868F870000F08080868F87000080
8080868F870000808080868970"):: CALL CHAR(131,"000000000
0609070")
110 CALL CHAR(132,"0000120C483020400000221C0810200000201
0201030200000003CFF"):: CALL CHAR(136,"000000FF3C")
120 CALL CLEAR :: S$="GFEDCBA" :: CALL CHAR(45,"00000000
FF"):: A$=RPT$(S$,3):: FOR R=2 TO 22 STEP 2 :: IF R=12 T
HEN 130 :: DISPLAY AT(R,1):RPT$("-",28)
130 NEXT R :: CALL CHAR(98,"0020202834242830")
140 FOR R=1 TO 21 :: DISPLAY AT(R,1):SEG$(A$,R,1);::
NEXT R
150 DATA 127,127,128,128,129,129,130,130,131,131
160 DATA 1/16,1/8,1/4,1/2,1/1
170 FOR R=1 TO 20 STEP 2 :: READ N :: DISPLAY AT(R,15):C
HR$(N);:: NEXT R :: FOR R=3 TO 19 STEP 4 :: DISPLAY AT(R
,16):".";:: NEXT R
180 C=132 :: FOR R=1 TO 17 STEP 4 :: DISPLAY AT(R,17):CH
R$(C);:: C=C+1 :: NEXT R
190 FOR R=1 TO 17 STEP 4 :: READ M$ :: DISPLAY AT(R,20):
M$;:: NEXT R
200 DATA 35,33,32,30,28,27,25,23,21,20,18,16,15,13,11,9,
8,6,4,3,1
210 FOR R=1 TO 21 :: READ N :: N$=N$&CHR$(N):: DISPLAY A
T(R,6):STR$(N);:: NEXT R
220 G$="b" :: Z=-1 :: GOSUB
320 :: IF F=0 THEN 230 ELSE GOSUB 330 :: GOTO 240
230 G$="#" :: Z=1 :: GOSUB 320 :: IF F<>0 THEN GOSUB 330
240 DISPLAY AT(24,1):"Shortest note? 1/" :: ACCEPT AT(24
,18)VALIDATE("12468")SIZE(2)BEEP:L :: T$="1/"&STR$(L)::
RESTORE 160 :: FOR J=1 TO 5 :: READ L$ :: IF L$=T$ THEN
260
250 NEXT J :: GOTO 240
260 DISPLAY AT(24,1):"Is it dotted? Y/N" :: ACCEPT AT(24
,19)VALIDATE("YN")SIZE(1):D$ :: D=1-(D$="Y")
270 T=-3+J*4
280 FOR R=T TO 19 STEP 4 :: DISPLAY AT(R,11):STR$(D);::
DISPLAY AT(R+2,11):STR$(D*1.5);:: D=D*2 :: NEXT R
290 GOTO 360
300 FOR R=1 TO 20 STEP 2 :: READ N :: DISPLAY AT(R,15):C
HR$(N);:: NEXT N
310 GOTO 310
320 DISPLAY AT(24,1):"How many "&G$&" on upper scale?" :
: ACCEPT AT(24,28)VALIDATE("01234567")SIZE(1)BEEP:F :: R
ETURN
330 Y$="" :: FOR J=1 TO F :: DISPLAY AT(24,1):"On which
letter?"
340 ACCEPT AT(24,18)VALIDATE(S$)SIZE(1)BEEP:L$ :: IF POS
(Y$,L$,1)<>0 THEN 340 ELSE Y
$=Y$&L$
350 S=1 :: FOR K=1 TO 3 :: P=POS(A$,L$,S):: DISPLAY AT(P
,2):G$;:: DISPLAY AT(P,6):STR$(ASC(SEG$(N$,P,1))+Z);:: S
=P+1 :: NEXT K :: NEXT J :: RETURN
360 OPEN #1:"PIO" :: FOR R=1 TO 22 :: FOR C=3 TO 30 :: C
ALL GCHAR(R,C,G):: CALL HCHAR(R,C,30):: R$=R$&CHR$(G)::
NEXT C :: PRINT #1:R$ :: R$="" :: NEXT R :: STOP
```

Get yourself a piece of sheet music and compare it to the screen display from that program. You will see that music is written on two sets of 5 lines. The upper set is marked at the left end with something like a fancy script capital S; it is used to write the higher notes, including the melody, which a pianist plays with the right hand. The lower set, marked with a sort of a backward C, contains the low notes played with the left hand. Your sheet music probably has a wide space between the sets, to make room for the lyrics, but there are really only three notes between them.

The screen display shows letters at the left, which are not on the sheet music. Those are the names of the notes, which we will have to refer to a couple of times to get started; observe that the notes are named A through G and then repeated.

The numbers along the left side are the numbers you will key in to play those notes. However, the screen display is set up in the key of C, which is played entirely on the piano white keys. The sheet music you want to program from may be in a different key, so - The computer is asking you how many there are of something that looks like a squashed lower case b - I guess that is why they call it a flat? It means that the note will be played a bit lower, on the black key just left of the white key - and we will program it one number lower. So, look next to that capital S and see how many flats there are. If none, type 0. Otherwise, the computer will ask which letters they are next to. Type them in, one at a time, and presto - the computer will put them on the staff and adjust the numbers accordingly.

If there were no flats, the computer will want to know if there are any sharps - those are what you get by typing a shift 3 on the keyboard, and they mean that the note is played on the black key above the white key, and is programmed one number higher.

Now, the computer needs some information in order to help you set up the length of your notes - how long they are sounded. The various notes are depicted at the right. A 1/16 note is a little black egg with a stem (it may go up or down, makes no difference) and two flags on the stem. A 1/8 has only one flag and a 1/4 note has none. A 1/2 note is a hollow egg with a stem and a whole note has no stem.

Those little doodads to the right of the notes are rests, used to indicate a silent pause of the same length as that note - more on that later. Look through your sheet music and find the shortest note. Tell the computer. It will want to know if any of those shortest notes are dotted - have a little dot to their right, as the screen display shows. A dotted note is played half again as long as normal. Presto again, the computer will show you the duration number to key in for each note. Then, if you have a printer attached, it will print out an Extended Basic screen dump of that screen - you will have to squash your own b's and sketch in the notes and rests.

If your software library contains an assembly screen dump, delete that last program line and put in a CALL INIT, CALL LOAD and CALL LINK to get a better printout - or ask me for it. If you do not have a printer, why not copy those numbers right onto the corresponding lines and spaces on your sheet music, and number some of the notes. Now we are ready to make music! Let's keep it simple at first, just a single note melody - and I hope you picked a simple piece of music. Clear the TI's brain with NEW, then merge in that line 100 scale from part 1 by MERGE DSK1.SCALE . In the same way, merge in one of those line 1000 CALL SOUND routines. Put in a temporary stopper line 999 STOP, and a line 110 D=200 to set the duration.

The melody is almost always on the upper set of 5 lines. If a note has 2 or 3 eggs on its stem, as they usually do, the upper one is the melody note - we will get into harmony later. Start with line 110. Check your chart to see what number denotes the length of the first note - maybe 2, if so key in T=2 :: Then check to see what number applies to the position of the upper egg of that note. Maybe 22, so key in A=22 :: GOSUB 1000 Enter RUN, and if you have done everything correctly, you will hear the note. You might decide already that you want to change that 200 in line 110.

Now for the second note. If it is of the same length as the first, you do not have to type anything - that is what makes this shorthand method so quick and easy. If the note position is also the same, you do not key that in either - just another GOSUB 1000.

If you have EZ-KEYS or another "hot keys" program, you can program a control key to put in the GOSUB 1000 with just one keypress - wish I had thought of that when I was programming music by the diskfull! So keep plugging along, keying in durations and notes. After every half dozen notes or so, type RUN to see if

everything sounds OK so far - it is easier to catch errors before they are too far back in the music.

You can get up to 5 screen lines on one line number, but you might better stick to 3 lines. You will note that the sets of notes are divided by vertical bars. You might program the notes between bars on a separate line, then add a ! followed by the words of the song that go with those notes - I find that a very good way to track down sour notes.

Regarding those bars - it might help you sometime to know this. At the beginning of the music, right after the big script S and the flats and sharps, you will see something like a 3 over a 4, or a 4 over a 4, or whatever - but often a symbol such as a barred C is used instead. A 3 over a 4, for instance, means that the notes between two of those bars will add up to 3/4 - might be three quarter notes, or two eighth notes and two quarter notes, or whatever, but they will add up to 3/4. Sometimes the very first notes will add up short, but in that case the very last ones will make up the difference.

The notes between those two bars make up a bar of music, and the emphasis is on the first note - for instance, that 3/4 is the 1-2-3, 1-2-3 beat of waltz time. While you are keying in that music, you might come to one of those rests. You can just key in its T= value and then A=0 for a silent note. However, computer notes stop so abruptly that somehow a rest just does not sound right, so I often just use the previous note instead.

You may come across one of those flat or sharp symbols next to a note in the music. Give the note a number 1 lower if a flat, one higher if a sharp, and the same for any subsequent occurrences of that note, until you find next to it a symbol that looks like the sharp sign with half its legs knocked off; that means to go back to normal. You might also come across that symbol to tell you to play a normally flat or sharp note as if it was not.

I think that covers all that you absolutely have to know for now, and I have horrified all serious students of music just about enough. There are all kinds of other squiggles on the sheet music but usually they are not essential in programming music. There is one other time-saving shortcut that I should tell you about right now. Most music consists at least partly of musical phrases, of a series of notes, which are repeated two or more times within a melody. So, the first thing you should do before you start programming a song is to search through the music for such phrases.

If you find one, of more than a few notes, that is repeated elsewhere - and make sure it is repeated exactly the same - mark it off each place it occurs and label it 500. If you find a second repeating phrase, label it 600, and so on. Then, when you start programming, start with line 500, key in that series of notes first, and end it with RETURN. If you have another phrase, put it in lines starting with 600, again ending with RETURN.

Now, start programming from the beginning of the song in line 120, but when you come to one of those phrases, just put in GOSUB 500 - the program will jump to that line number, play those notes, and come right back to where it was.

In Part 3, we will get into programming in 3-part harmony, bass notes, auto-chording, and all kinds of things. *O*

continued from page 11

address which specifies the S type address as an accumulator, "C" (R12), since the first two bits of the six are 00.

The TMS9900 processor always reads the source address even if it is going to ignore its contents and over-write it with something else, in this case zeros.

CRU Addressing

Although Ac"C" can be used as an accumulator in the usual way, it also has a special function for the TMS9900 in that it defines the CRU base address. Instructions 1D, 1E and 1F use it to add the offset (doubled) to the base address to locate a CRU bit. In this case, the base address is set to zero and this selects "RAMBLK" level of the TI99/4A, which is active low only if A3, A4 and A5 are low. Figure 2 shows the logic diagrams (part) involved.

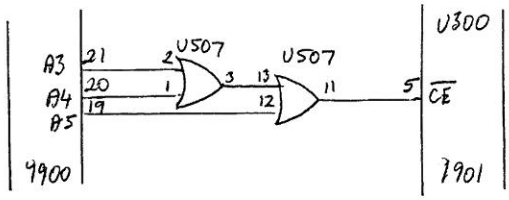

Figure 2. TI99/4A Schematic Diagram
Extracts from sheets 2 and 3
Note: U507 is a 74LS32 chip, a quad OR gate

Output levels from U300 strobe matrix lines of the console keyboard and pressed keys connect interrupt ports of U300 to these levels. Thus the stage is set for detection of a key press.

More in a later contribution.

9900 "MacroT" De-compiler

F	IORB		0B	ROT	
E	IOR		0A	ASHL	c,s
D	MOVB		09	LSHL	
C	MOVE		08	ASHR	
B	ADDB				
A	ADD	D,S			
9	COMPB		074	MOVMS	
8	COMP		070	SETOS	
7	SUBB		06C	MOVSS	
6	SUB		068	JSP	
5	ANDCB		064	STS	
4	ANDC		060	SOS	
			05C	ATS	S
3C	DIV	d,S	058	AOS	
38	MUL		054	SETCS	
			050	MOVNS	
34	STCR	c,S	04C	SETZS	
30	LDCR		048	XCT	
			044	JUMPA	
2C	EXOP	d,S	040	LWPJ	
28	XOR		03E	LREX*	
24	TZC	d,S	03C	CKOF*	
20	TOC		03A	CKON*	nil
			038	RRWP	
1F	TBIT		036	RESET*	
1E	SBZ	off.	034	IDLE*	
1D	SBO				
			030	LIMI	.+2
1C	SKIPOP		02E	LWPI	
1B	SKIPUG				
1A	SKIPUL		02C	STSR	s
19	SKIPNO		02A	STWP	
18	SKIPC				
17	SKIPNC		028	CI	
16	SKIPNE	c	026	IORI	
15	SKIPG		024	ANDI	s,.+2
14	SKIPUGE		022	ADDI	
13	SKIPE		020	MOVEI	
12	SKIPULE				
11	SKIPL		. Address of current instruction		
10	SKIPA				

continued on page 20

Sorting part 6

by Ron Brubaker, USA

This chapter describes a sorting algorithm called Quicksort II. It is one of the finest sorting methods. It is slightly faster than the shell sort for longer lists but due to its complexity it loses the race on shorter sorts to the more straight forward shell sort.

The Quicksort II algorithm sorts in a very simple manner. First, it chooses a value near the centre of the list, which is not necessarily near the middle value. It then looks at the elements to its left starting with the first. If the first is smaller than the value chosen it moves to the next value. This continues until a larger value is found. Then it looks at the vakue at the right end of the list. If this value is larger than the value originally chosen it moves to the next-to-last segment. Whenever it has found an element to the left of the middle value that is too large, and an element to the right of the middle value that is too small, it swaps them. Inevitably, it will end this phase of its work with all of the smaller values to the left of the arbitrary middle value and all of the larger values to the right. Then it divides the list into parts by storing the indices of the first and last elements of the piece containing the larger values in two stack arrays L() and R(). It then chooses a value in the middle of the remaining piece, the one with the smaller values, and begins the process of sorting them into two parts again. Ultimately, it reaches the point, where, given a list of three numbers, if you put the smallest one on the left and the largest one on the right, they are sorted. By this time it has a bunch of unfinished business on the stacks where it has sorted the ends of the pieces of the list it has not finished. These are retrieved one at a time and are resolved, often causing more items to be placed on the stack in the process.

Seems like a weird way to do business, but the proof is in the results and this thing is fast! Try it. Be careful though, if you make a mistake in the program it may never get done and you will have a lot of fun finding your error.

Each of the following is a full working program in which the sorting routine is used as a subroutine to facilitate incorporation into your favourite program.

PROGRAM 1

```
100 !************************
110 !PROGRAM TO DEMONSTRATE
         QUICKSORT II
120 !************************
130 !Based on an article on
     sorting in the Sept 81
     issue of Interface Age
        by Gene Cotton.
140 !************************
      NUMERIC -- VERSION
     ************************
150 !VARIABLE KEY
160 ! S1=CURRENT STACK VALUE
170 !L()=STACK VALUE FOR LEFT END OF LIST
180 !R()=STACK VALUE FOR RIGHT END OF LIST
190 ! L1=LEFT END OF CURRENT SEGMENT
200 ! R1=RIGHT END OF CURRENT SEGMENT
210 ! L2=LEFT POINTER IN CURRENT SEGMENT
220 ! R2=RIGHT POINTER IN CURRENT SEGMENT
230 !  X=VALUE OF MID POSITION VARIABLE IN
         CURRENT SEGMENT
240 !  T=TEMPORARY VARIABLE TO HOLD VALUE
         DURING SWAP
250 !  N=LENGTH ARRAY(A)
260 !NOTE, THIS PROGRAM ASSUMES THAT THE ARRAY
          TO BE SORTED IS A().
270 !THE STACK ARRAYS L() AND R() SHOULD BE
280 !DIMENSIONED APPROXIMATELY 1/3 AS LARGE AS A()
290 !************************
        SORTING ROUTINE
    ************************
300 DIM A(100),L(33),R(33)
310 IMAGE ####
320 GOTO 640
330 S1=1
340 L(1)=1
350 R(1)=N
360 L1=L(S1)
370 R1=R(S1)
380 S1=S1-1
390 L2=L1
400 R2=R1
410 X=A(INT((L1+R1)/2))
420 IF A(L2)>=X THEN 450
430 L2=L2+1
440 GOTO 420
450 IF X>=A(R2)THEN 480
460 R2=R2-1
470 GOTO 450
480 IF L2>R2 THEN 540
490 T=A(L2)
500 A(L2)=A(R2)
510 A(R2)=T
520 L2=L2+1
530 R2=R2-1
540 IF L2<=R2 THEN 420
550 IF L2>=R1 THEN 590
560 S1=S1+1
570 L(S1)=L2
580 R(S1)=R1
590 R1=R2
600 IF L1<R1 THEN 390
610 IF S1>0 THEN 360
620 RETURN
630 !************************
      GENERATE A LIST OF
        RANDOM NUMBERS
    ************************
640 RANDOMIZE
650 PRINT "HOW MANY NUMBERS DO YOU WANT";
660 INPUT N
670 PRINT
680 FOR I=1 TO N
690 A(I)=INT(100*RND)+1
700 PRINT USING 310:A(I);
710 NEXT I
720 PRINT
730 !************************
      CALL SORT ROUTINE
    ************************
740 GOSUB 330
750 !************************
     ROUTINE TO PRINT THE
          SORTED LIST
    ************************
760 PRINT
770 FOR I=1 TO N
780 PRINT USING 310:A(I);
790 NEXT I
800 PRINT
810 END
```

PROGRAM 2

```
100 !************************
110 !PROGRAM TO DEMONSTRATE
         QUICKSORT II
120 !************************
130 !Based on an article on
     sorting in the Sept 81
     issue of Interface Age
        by Gene Cotton.
140 !************************
      ALPHABETIC - VERSION
     ************************
150 !VARIABLE KEY
160 ! S1=CURRENT STACK VALUE
170 !L()=STACK VALUE FOR LEFT END OF LIST
180 !R()=STACK VALUE FOR RIGHT END OF LIST
190 ! L1=LEFT END OF CURRENT SEGMENT
200 ! R1=RIGHT END OF CURRENT SEGMENT
210 ! L2=LEFT POINTER IN CURRENT SEGMENT
```

```
220 ! R2=RIGHT POINTER IN CURRENT SEGMENT
230 ! X$=VALUE OF MID POSITION VARIABLE IN
        CURRENT SEGMENT
240 ! T$=TEMPORARY VARIABLE TO HOLD VALUE
        DURING SWAP
250 !  N=LENGTH ARRAY A$()
260 !NOTE, THIS PROGRAM ASSUMES THAT THE ARRAY
         TO BE SORTED IS A$( ).
270 !THE STACK ARRAYS L() SHOULD BE DIMENSIONED
     APPROXIMATELY 1/3 AS LARGE AS A$().
280 !THE STACK ARRAY R() SHOULD BE DIMENSIONED
     AS LARGE AS A$().
290 !************************
       SORTING ROUTINE
     ************************
300 DIM A$(15),L(15),R(15)
310 GOTO 630
320 S1=1
330 L(1)=1
340 R(1)=N
350 L1=L(S1)
360 R1=R(S1)
370 S1=S1-1
380 L2=L1
390 R2=R1
400 X$=A$(INT((L1+R1)/2))
410 IF A$(L2)>=X$ THEN 440
420 L2=L2+1
430 GOTO 410
440 IF X$>=A$(R2)THEN 470
450 R2=R2-1
460 GOTO 440
470 IF L2>R2 THEN 530
480 T$=A$(L2)
490 A$(L2)=A$(R2)
500 A$(R2)=T$
510 L2=L2+1
520 R2=R2-1
530 IF L2<=R2 THEN 410
540 IF L2>=R1 THEN 580
550 S1=S1+1
560 L(S1)=L2
570 R(S1)=R1
580 R1=R2
590 IF L1<R1 THEN 380
600 IF S1>0 THEN 350
610 RETURN
620 !************************
      READ A LIST OF RANDOM
          ORDERED NAMES
     ************************
630 READ N
640 FOR I=1 TO N
650 READ A$(I),R(I)
660 NEXT I
670 DATA 15
680 DATA "WASHINGTON, GEORGE",1,"JEFFERSON, THOMAS",3
690 DATA "FORD, GERALD",37,"KENNEDY, JOHN",34
700 DATA "FILLMORE, MILLARD",13,"ARTHUR, CHESTER",21
710 DATA "ADAMS,JOHN Q",6,"LINCOLN, ABRAHAM",16
720 DATA "ROOSEVELT, FRANKLIN",31,"REAGAN, RONALD",39
730 DATA "CARTER, JAMES",38,"WILSON, WOODROW",27
740 DATA "MONROE, JAMES",5,"ROOSEVELT, THEODORE",25
750 DATA "ADAMS, JOHN",2
760 !************************
       CALL SORT ROUTINE
     ************************
770 GOSUB 320
780 !************************
      ROUTINE TO PRINT THE
          SORTED LIST
     ************************
790 PRINT
800 FOR I=1 TO N
810 PRINT A$(I)
820 NEXT I
830 PRINT
840 END
```

PROGRAM 3

```
100 !************************
110 !PROGRAM TO DEMONSTRATE
        QUICKSORT II
120 !************************
130 !Based on an article on
     sorting in the Sept 81
    issue of Interface Age
         by Gene Cotton
140 !************************
       NUMERIC -- VERSION
          WITH POINTER
     ************************
150 !VARIABLE KEY
160 !  S1=Current stack value
170 !L( )=Stack value for left end of list
180 !R( )=Stack value for right end of list
190 !  L1=Left end of current segment
200 !  R1=Right end of current segment
210 !  L2=Left pointer in current segment
220 !  R2=Right pointer in current segment
230 !   X=Value of mid position variable in
          current segment
240 !   T=Temporary variable to hold value
          during swap
250 !   N=Length Array(A)
260 !Note, This program assumes that the array
          to be sorted is A( ).
270 !The stack arrays L( ) and R( ) should be
280 !dimensioned approximately 1/3 as large as A()
290 !************************
       SORTING ROUTINE
     ************************
300 DIM A(100),P(100),L(33),R(33)
310 IMAGE ####
320 GOTO 640
330 S1=1
340 L(1)=1
350 R(1)=N
360 L1=L(S1)
370 R1=R(S1)
380 S1=S1-1
390 L2=L1
400 R2=R1
410 X=A(P(INT((L1+R1)/2)))
420 IF A(P(L2))>=X THEN 450
430 L2=L2+1
440 GOTO 420
450 IF X>=A(P(R2))THEN 480
460 R2=R2-1
470 GOTO 450
480 IF L2>R2 THEN 540
490 T=P(L2)
500 P(L2)=P(R2)
510 P(R2)=T
520 L2=L2+1
530 R2=R2-1
540 IF L2<=R2 THEN 420
550 IF L2>=R1 THEN 590
560 S1=S1+1
570 L(S1)=L2
580 R(S1)=R1
590 R1=R2
600 IF L1<R1 THEN 390
610 IF S1>0 THEN 360
620 RETURN
630 !************************
      GENERATE A LIST OF
        RANDOM NUMBERS
     ************************
640 RANDOMIZE
650 PRINT "HOW MANY NUMBERS DO YOU WANT";
660 INPUT N
670 PRINT
680 FOR I=1 TO N
690 P(I)=I :: A(I)=INT(100*RND)+1
700 PRINT USING 310:A(I);:: NEXT I
710 PRINT
720 !************************
       CALL SORT ROUTINE
     ************************
730 GOSUB 330
740 !************************
      ROUTINE TO PRINT THE
          SORTED LIST
     ************************
750 PRINT
760 FOR I=1 TO N :: PRINT USING 310:A(P(I));:: NEXT I
```

PROGRAM 4

```
100 !***********************
110 !PROGRAM TO DEMONSTRATE
        QUICKSORT II
120 !***********************
130 !Based on an article on
     sorting in the Sept 81
     issue of Interface Age
        by Gene Cotton
140 !***********************
      ALPHABETIC - VERSION
         WITH POINTER
     ***********************
150 !VARIABLE KEY
160 !  S1=Current stack value
170 !L( )=Stack value for left end of list
180 !R( )=Stack value for right end of list
190 !  L1=Left end of current segment
200 !  R1=Right end of current segment
210 !  L2=Left pointer in current segment
220 !  R2=Right pointer in current segment
230 !  X$=Value of mid position variable in
          current segment
240 !  T$=Temporary variable to hold value
          during swap
250 !P( )=Sorting pointer array
260 !   N=Length Array A$()
270 !Note, This program assumes that the array
          to be sorted is A$( ).
280 !The stack array L( ) should be dimensioned
     approximately 1/3 as large as A$().
290 !The stack aray R() should be dimensioned
     as large as  A$().
300 !***********************
        SORTING ROUTINE
     ***********************
310 DIM A$(15),L(15),R(15),P(15)
320 GOTO 640
330 S1=1
340 L(1)=1
350 R(1)=N
360 L1=L(S1)
370 R1=R(S1)
380 S1=S1-1
390 L2=L1
400 R2=R1
410 X$=A$(P(INT((L1+R1)/2)))
420 IF A$(P(L2))>=X$ THEN 450
430 L2=L2+1
440 GOTO 420
450 IF X$>=A$(P(R2))THEN 480
460 R2=R2-1
470 GOTO 450
480 IF L2>R2 THEN 540
490 T=P(L2)
500 P(L2)=P(R2)
510 P(R2)=T
520 L2=L2+1
530 R2=R2-1
540 IF L2<=R2 THEN 420
550 IF L2>=R1 THEN 590
560 S1=S1+1
570 L(S1)=L2
580 R(S1)=R1
590 R1=R2
600 IF L1<R1 THEN 390
610 IF S1>0 THEN 360
620 RETURN
630 !***********************
      READ A LIST OF RANDOM
          ORDERED  NAMES
     ***********************
640 READ N
650 FOR I=1 TO N
660 P(I)=I
670 READ A$(I),R(I)
680 NEXT I
690 DATA 15
700 DATA "WASHINGTON, GEORGE",1,"JEFFERSON,THOMAS",3
710 DATA "FORD, GERALD",37,"KENNEDY, JOHN",34
720 DATA "FILLMORE, MILLARD",13,"ARTHUR, CHESTER",21
730 DATA "ADAMS, JOHN Q",6,"LINCOLN, ABRAHAM",16
740 DATA "ROOSEVELT, FRANKLIN",31,"REAGAN, RONALD",39
750 DATA "CARTER, JAMES",38,"WILSON, WOODROW",27
760 DATA "MONROE, JAMES",5,"ROOSEVELT, THEODORE",25
770 DATA "ADAMS, JOHN",2
780 !***********************
       CALL SORT ROUTINE
     ***********************
790 GOSUB 330
800 !***********************
      ROUTINE TO PRINT THE
          SORTED LIST
     ***********************
810 PRINT
820 FOR I=1 TO N
830 PRINT USING "  ##  ":R(P(I));
840 PRINT A$(P(I))
850 NEXT I
860 PRINT
870 END
```

continued from page 10

The final procedure is very similar to the previous case except that the number of characters to be examined causes a few changes. Another integer is declared (but not used) and the loop exit test is now determined by the value of n which is decremented in each loop. This means that at most n characters are examined, which is what the procedure is supposed to do, while a test must be included for the possibility that both strings end prematurely. Only one test is required as if only one string ends prematurely, the strings will be different at that point and the other return will be taken. Notice that the logical test for equality is two "=" characters. The last two statements are to do the nth and last test. there is also a test on n before the loop starts to make sure that the loop can finish. The result returned if n is less than 1 is equality and no comparison is made.

I hope you have followed my explanations here. Please contact me if you want more on any part of the explanations. Next month I will give you a main program to test these procedures and look at some of the other procedures in this library.

continued from page 21

```
[If b1=b2 this is the result]
LOOP     Z 0
SWAP     0 Z
DROP     0        [false flag]
```

NOTES On SORT$ etc

Note 1. I is adr2 since we set the loop index to be adr2.

Note 2. Subtracting b2 from b1 results in a value which is used as a flag (f) by IF,thus:

f is 0 (false) if b1=b2
f is <0 (true) if b1<b2
f is >0 (true) if b1>b2

Any nonzero value is a true flag.

Note 3. The sign of the flag f is be used during an alphabetical sort to decide which character has the larger ASCII value, since the ASCII values increase uniformly from 0 to 9, A to Z and a to z.

Note 4. If the comparison results in a non-zero value because the characters compared are not equal, the words after the IF are executed and LEAVE causes exit from the DO loop, because if any character in one string differs from the other, the strings are not equal and our concern is only which has the larger ASCII value. If the result is zero (ie the characters do match) LOOP causes a return to the word following DO and another pair of characters are selected for comparison. When the required count (u) of characters has been tested and no inequality found, the loop is exited. The SWAP DROP after the loop gets rid of the garbage on the stack leaving the +/-1 or 0 flag.

How to Use Arrays

by Andy Frueh, OH USA

I have considered myself decent in Extended BASIC programming for a few years. However, one thing eluded me. I could not figure out what "dimensional array" meant. Obviously, it was a powerful thing, but I could not quite figure out how to use them. Why would I want to?

For those who are just starting out, strap yourself in. This could be a bumpy ride.

How can I explain arrays? Well, I suppose the same way I explained them to myself when I finally understood them. An array is nothing more than a chart in the computer's memory. It has a certain number of rows and a certain number of columns.

I assume most of us understand this BASIC command. 10 A=5 This sets the variable A to the value of 5. The next line could say 20 PRINT A-5 When that program runs, your answer would be 0. A=5 and if you subtract 5 from 5 you get zero. Simple!

An array is a slightly more complicated set of variables.

10 DIM A(5) Since there is one number in parentheses, it is a "one dimensional" array. You could say it has 1 row and 5 columns. Your next line could say 20 A(0)=1 :: A(1)=2 :: A(2)=4 :: A(3)=8 :: A(4)=16 HANG ON! Why did I start with 0? Because to the computer, counting starts with 0. I realize that this confuses some people. If you would rather start counting with 1, place this statement in the beginning of your program... OPTION BASE 1 What the heck does THAT mean? Well, you can use OPTION BASE 0 or OPTION BASE 1 in your programs. You use OPTION BASE only ONCE. This tells the computer whether you are going to start counting with 0 or 1. If we had OPTION BASE 1 as line 5 in our little program, the numbers in parentheses in line 20 would each have to be increased by one, so they would be values from 1-5.

In our program (let's use OPTION BASE 1), we have set up a one dimensional array, with 5 columns or elements. You cannot get a picture in your mind? Well, basically, this is what we have:

```
                ------------
           ()   |1|2|3|4|5 |
Array           ------------
  A      Value: |1|2|4|8|16|
                ------------
```

We can use more than one row. Here is another little program and it's chart.

```
5 OPTION BASE 1
10 DIM A(2,2)
20 A(1,1)=1 :: A(1,2)=2 :: A(2,1)=3 :: A(2,2)=4
```

```
           ---------
       1   | 1 | 2 |
Array  ,   ---------
  A    2   | 3 | 4 |
           ---------
```

You may have 7 dimensional arrays in Extended BASIC. For example, you might need to write an address program. You decide that each piece of information will be stored separately. For example, although the street address is asked for on one line, the number, street, and apartment number are all going to be stored in a different variable. You could do this:

```
5 OPTION BASE 1
10 DIM A$(3,3,1,1,1,7,2)
```

The first set of three would be first name, middle initial and last name. Three pieces of information are in the FIRST DIMENSION of the array. The next set of three would be the street number, street name, and apartment number. Next, the user would enter the city. This is the only piece of information in this dimension (dimension 3). The 4th dimension would contain the state, and the fifth would be the zip code. Dimension 6 could be used to store 7 pieces of information on the person at that address. Profession, where they work, etc. The last and 7th dimension has two pieces of information to it. The first would be the area code, and the second, the phone number.

How about a "practical" illustration? The program below will load in two full screens of a text file. We have 28 columns available to us normally in Extended BASIC, using the PRINT routine. There are 80 columns in a standard text file (D/V 80). Some quick Maths tells us that we will need 3 screen lines to display 1 text file line. OK, some more quick Maths. There are 24 lines on a screen, so we can only put 8 lines of the text file on one screen. If our program loads in TWO screens, we would need a total of 16 text file lines. Our math is now out of the way...we know what we need to do.

The program below will load the text into the array TEXT$(). It is two dimensional. Remember that means that there will be two major pieces of data. These two pieces will be the number of screens and the lines in each screen. For example, in the program below, I use DIM TEXT$(2,8). The way I am using it is like this: the two represents the "pages" or number of screens. In TWO screens, I need 8 LINES.

```
10 OPTION BASE 1
20 DIM TEXT$(2,8)
30 CALL CLEAR::INPUT "Text file?":A$
40 OPEN #1:A$,INPUT
50 FOR S=1 TO 2::FOR L=1 TO 8
60 INPUT #1:TEXT$(S,L)
70 NEXT L::NEXT S::S=0
80 CALL CLEAR::S=S+1
90 FOR L=1 TO 8
100 PRINT TEXT$(S,L)::NEXT L
110 CALL KEY (0,K,ST)::IF ST=0 THEN 110 ELSE IF S<2 THEN
80 ELSE END
```

You may get an error message if the text file is less than a total of 16 lines. Watch out for this! You will need to press a key between pages. If all 8 lines are not completely filled, neither will the screens be filled. o

continued from page 8

Run times depended on line 30:

Line 30	Run Time
GOSUB 50	26.9 seconds
GOSUB 2060	12.6 seconds
CALL A	15.6 seconds

GOSUB 50 took longer than GOSUB 2060 as the computer had to wade through 202 lines to find line 50 in the line number table versus one line to find line 2060.

Clearly, in long programs, put your frequently used subroutines and groups of code at the end of your program.

SUMMARY

For other ideas read the September, 1984 issue of Millers Graphics' "The Smart Programmer" and a November, 1983 article in "99'er Home Computer Magazine" by John Dow, "Squeezing the most out of TI Basic".

Experiment. You will find other time savers. If you run some of the programs in this article you will probably get slightly different times because you may be more accurate with your stopwatch and your programs may differ slightly from mine.

I hope that these ideas will help you write and refine your Extended Basic Programs. o

TI-Base Tutorial #16

by Martin Smoley, NorthCoast 99'ers, USA


```
* HAPPY NEW YEAR *

*   Find Past Members  FPM5/C
CLOSE ALL
LOCAL FDT C 5
LOCAL LDT C 5
*  SET TALK OFF
   SET RECNUM OFF
   SET HEADING OFF
PRINT (f)
    USE TNAMES
    CLEAR
WRITE 8,1," Enter Date as YY/MM"
WRITE 10,1," Enter First Date: 88/05"
WRITE 14,1," Enter Last Date: 90/02"
READSTRING 11,20,FDT
READSTRING 15,19,LDT
 WHILE .NOT. (EOF)
PRINT NM,FDT,XP,LDT,ST,SA,ZP
 IF (((XP>FDT).AND.(XP<LDT)).OR.;
    ((XP=FDT).OR.(XP=LDT))).AND.;
    (ST="OH").AND.(SA<>"No Newsletter");
    .AND.(ZP<>" Zip ")
  PRINT (LF)
  PRINT
  PRINT (LF)
  ENDIF
   MOVE
 ENDWHILE
CLOSE ALL
   SET TALK ON
   SET RECNUM ON
   SET HEADING ON
RETURN Copyright Martin A. Smoley 1990
```

I cannot believe it is 1990 already. I have not made any New Year resolutions, but if I had known I was going to live this long, I would have taken better care of myself.

At this point I really believe that I have covered all the functions of TI-Base. Therefore, I intend to ramble along in a haphazard manner and build my tutorials on things which I am doing during the month the tutorial is due. If possible I will link my tutorials to the previous month and relate the work to TNAMES, or some other small database to give you a working model of the idea I am presenting.

Last month I gave myself the task of pulling the names of past members out of the NorthCoast membership list for the purpose of sending them a letter. We are having a swap meet at the February meeting and we wanted to give ex-members the chance to sell their TI99/4A components (current members should read about this item in the Newsletter). This was actually an easy task to start with, but I managed to complicate it considerably, as usual. The first thing I did was to print out a complete old membership list. As I backup my disks I save the old disks to a storage box with the most current at the front and the oldest at the back. I still have disks from 1983. This means that I can back up two or three disks and find all the past NorthCoast members. By writing a command file that looks through these data bases using a date range and a couple other parameters, I can extract the names I want. I expanded TNAMES and edited it so it would give me some problems. That may sound strange, but it is exactly correct. If you want to do a job with a specialized command file, you need a small data base to test the command file. The data base should contain data with a wide variety of dates, names, addresses, etc. to thoroughly test your command file. I look through the data base I wish to use and pull out items I think will give me problems. Then I incorporate these problems into my small test data base so I can iron them out before I try it on the important data. It may sound like a lot of effort to keep changing TNAMES, but TNAMES is one of my most productive debugging tools.

Since the last listing of TNAMES I have added record 0 (Zero). You will notice that it is really a description of the fields. The CT field contains the word City and the last date the data base was modified. You should also notice that many of the fields in record 0 start with a blank space. In most cases that blank space will return record 0 to the top of the list when the data base is sorted on those fields. Using record 0 gives me a nice printout with no added effort when I am printing the data base, and I have the Heading and Recnum turned off. There are many ways to keep record 0 from being mixed in with the rest of the data when you do not want it there. You can see one of my methods for accomplishing this task at the very end of the IF statement.

```
*  Find Past Members  FPM8/C
CLOSE ALL
    USE DSK1.TNAMES
 SET PRINTER=DSK1.PASTNMS
 SET RECNUM OFF
 SET HEADING OFF
PRINT (f)
PRINT
LOCAL FDT C 5
LOCAL LDT C 5
   SET TALK OFF
    CLEAR
WRITE 8,1," Enter Date as YY/MM"
WRITE 10,1," Enter First Date: 88/05"
WRITE 14,1," Enter Last Date: 90/02"
READSTRING 11,20,FDT
READSTRING 15,19,LDT
 WHILE .NOT. (EOF)
```

Listing of modified TNAMES database. 01/01/90

REC	NM	LN	FN	MI	SA	CT		ST	ZP	PH	XP	GP	ID
0008	0	Last Name	First Name	MI	Street Address	City	01/01/90	ST	Zip	Phone	XPDt	Group	ID-No.
0002	1	Aardvark	Grant	E.	9995 State Rt. 84	Geneva		OH	44014	1-465-9876	89/12	NOCO	0717851
0003	2	Aardvark	Willard	J.	No Newsletter			OH		1-465-7689	88/09	NOCO	0717852
0006	3	Jones	Jan	M.	456 Skytop Drive	Vale		CO	80012		90/08	NOCO	0321891
0005	4	Jones	Quincy	W.	37285 Burgandy Lane	Mentor-o	n-the-Lake	OH	44060	257-1029	90/03	NOCO	0820871
0000	5	Smoley	Martin	A.	6149 Bryson Drive	Mentor		OH	44060	216-257-1661	90/02	NOCO	0713831
0007	6	Vilario	Aulga	J.	9238 Townsend Road	St. Louis		MO	63174	314-868-8641	90/06	NOCO	0804891
0004	7	Vivannovitch	Elexxie	I.	111 E. 98th. St.	Cleveland		OH	91023	541-5415	88/05	NOCO	0712881
0001	8	Whitman	Raymond (Slim)	B.	2574 East 254th.	Eastlake		OH	44094	951-2345	89/09	NOCO	0921861

```
 IF (((XP>FDT).AND.(XP<LDT)).OR.;
    ((XP=FDT).OR.(XP=LDT))).AND.;
  (ST="OH").AND.(SA<>"No Newsletter");
    .AND.(ZP<>" Zip ")
DO DSK1.SV'PRNT
  ENDIF
   MOVE
 ENDWHILE
CLOSE ALL
   SET TALK ON
   SET RECNUM ON
   SET HEADING ON
 SET PRINTER=PIO.CR.LF
RETURN Copyright Martin A. Smoley 1990
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*                          SV'PRNT
SET PAGE=000
CLEAR
LOCAL TEMP C 40
WRITE 10,5,"Saving name to DV/80 file"
SET PRINTER=DSK1.PASTNMS
PRINT
SET PRINTER=RS232.CR.LF.DA=8
PRINT (E)
WRITE 10,5,"   Printing one Label    "
*
   REPLACE TEMP WITH "           ";
     ¦ "   Exp. Date: " ¦ XP
     PRINT TEMP
     PRINT (CR),(LF)
   REPLACE TEMP WITH TRIM(FN) ¦ " ";
           ¦ MI ¦ " " ¦ LN
     PRINT TEMP
     PRINT SA
   REPLACE TEMP WITH TRIM(CT) ¦ " ";
           ¦ ST ¦ ". " ¦ ZP
     PRINT TEMP
     PRINT (CR),(LF)
RETURN Copyright Martin A. Smoley 1990
```

Let us get to the command files. FPM5/C is only included to demonstrate my heavy use of PRINT statements in the testing of complicated clauses. I created FPM5/C as you see it except the IF statement was a simple "IF (XP>FDT).AND.(XP<LDT)". I ran the command file which printed the data it was using, for my verification. Then I added the rest of the IF requirements (one at a time) and ran the command file each time to see the results. You will notice that this particular IF statement has seven requirements before the PRINT commands within the IF-ENDIF will execute. This amount of programming power is also usable with the ;FOR clause which was discussed last week. After assuring myself that things were going the way I wanted, I moved along to the two command files you see on this page. I rearranged the top 10 lines of FPM8 so they will scroll up the screen before talk is set off. I might also place a (WAIT 5) just before CLEAR. This allows me a quick glimpse at what is going on in this command file when I use it four months from now. The important items would be, that I am using TNAMES and PRINTing to a file named PASTNMS, both on DSK1. The items for you to concentrate on at this point in the command file are SET PRINTER=DSK1.PASTNMS, PRINT (f) and PRINT. SET PRINTER=DSK1.PASTNMS tells TI-Base that until I make a change, send everything to be PRINTed to a DV/80 file on disk drive number 1. PRINT (f), sends one character to the DV/80 file, my command to set the printer to Condensed Mode. The next PRINT command tells TI-Base to PRINT the record it is pointing to, which in this case is record 0 (zero). This puts my headings at the top of PASTNMS. After the CLEAR, FPM8 asks you for the oldest allowable search date and the latest allowable search date. FPM8 should give you more instructions about entering these dates, but I did not want to waste the newsletter space. FDT should be the oldest date you want and LDT would be the newest or latest date. Note that in TNAMES, XP is not a date field. It is C)haracter type field. For this reason I put in the examples "88/05" and "90/02" to remind me of the type of input I must make. I am not sure, but I think this is the first time I am using user input for the execution of a search clause (IF) and the input must be exact or the IF statement will not find a match, or it might match everything. The IF clause says if XP is equal to or in between the items I entered (FDT and LDT) and ST equals OH and SA is not "No Newsletter" and ZP is not " Zip ", then DO DSK1.SV'PRNT. "TI-Base is capable of doing some pretty complicated stuff." If all these requirements are met, TI-Base will DO the command file SV'PRNT. SV'PRNT SETs the PAGE length to 000 then sends the complete record it has found in the IF clause to the DV/80 file with the first PRINT. The line that reSETs the PRINTER to DSK1.PASTNMS means nothing at this point, but it will be needed the next time SV'PRNT is run. I am saving the records to a DV/80 file so that I will have a complete Funnelweb list of all the people who will receive the letter. Now I need a label. The command SET PRINTER=RS232.CR.LF.DA=8 points TI-Base toward, and sets up, my printer, as you should recognize by now. PRINT (E) sets my printer to Emphasized Mode and the rest of SV'PRNT prints the label I want (hopefully). At the completion of the label, TI-Base RETURNs to FPM8 at the ENDIF statement. TI-Base then MOVEs to the next record in the Database and loops back up to the WHILE statement. If it has not hit the End Of File marker the statements in the WHILE loop will be executed. When the whole Database has been searched (one record at a time) for the records I want, the data base is closed and everything is reSET to my normal SETUP. I only use my RS232 for printing labels, so I reSET my PRINTER to PIO.CR.LF.

NOTE: Due to my use of the Formatter to print the text in these Tutorials, the spacing of SET statements may not be correct. The spacing in the command file listing should be correct and that is what you should follow.

TIP: If your command file has successfully opened a data base and you still get a "database not in use" error later in the command file, this probably means that you have made a mistake in the IF clause.

Continued Next Month.

continued from page 14

```
Notes: D    Full destination address, 6 bits
       d    Accumulator destination address, 4 bits
       S    Full source address, 6 bits
       s    Accumulator source address, 4 bits
       c    Count, 4 bits or 8 bits
       off. Offset from CRU address in Ac"C", doubled
Address Modes: 0    Ac, accumulator
               1    @Ac, accumulator indirect
              2(0) @.+2 symbolic memory
               2    @.+2(Ac) indexed memory
               3    @Ac add 1 or 2 to Ac,
                         auto-increment

* Not valid for TMS9900
```

Figure 1. Mnemonics

continued from page 6

fully commented version to allow you to learn how Jim programmed his calculator, and a stripped down version ready to run. The disk comes with detailed documentation

TC-1015 is entirely devoted to word processing utilities, written by Jim himself, as well as from other sources. The various programs can all be selected from the menu which loads automatically when XB is selected. The following utilities are available:

1 Carriage Return Adder
2 Reformatter+ V1.2
3 Tigercub Labeler V2.1
4 Tigercub Printall V.1.6
5 Printer Options
6 Text Reformatter
7 Program Relister
8 Text to Program Converter
9 28 Column Converter

Beginning Forth - part 16

by Earl Raguse, UGOC, CA USA

SIMPLE SORTING

I am about to deliver on the promise I made last time to show how to sort in Forth. The word today is SORT$ and it expects (adr1 adr2 cnt) on the stack. If you compare SORT$ to CSTR$ you will find great similarities between the words. In fact the only differences are the substitution of ROT for SWAP and the dropping of the 0= at the end of CSTR$.

The word SORT$ is demonstrated in Screens #51 through #55, the heart of the matter is really all on Screens #52 and #53. If you have any file called NAMELIST (equated to NL because it is used so often) and the variables N, TEST and TEMP and the prompting words on Screen #52, then the word SF (SortFile) will do a "bubble sort" on a file, in this case Screen #56, a dummy NameList file, generated for demo purposes - no copywrite infringement intended, Mother Goose. There is a display of the result. Screens #51 and part of #52 provide a means of creating this file using MAKEFILE which is equated to MF for ease of use. This is mostly by way of demonstration and can only create a file which looks like a phone number list. One could save and retrieve this file to and from disk using the techniques which were described in BFORTH #15.

MF (MakeFile) permits one to create a file on a screen using the very flexible Forth full screen Editor. The file, of course must be formatted properly according to the byte count assigned to each field. In this case 20 bytes for the name and 15 bytes for the phone number. Within a field, bytes (characters) may be anywhere. When you have properly entered the data on the screen, you can transfer it to NAMEFILE using LOADFILE equated to LF, and then PRINTFILE shortened to PF lets you display it to the screen.

If you should ever try to load more characters into a variable than was assigned for it in the ALLOT statement, Forth crashes in a very strange way and you have to reboot. I can attest to that from personal experience, which I will relate later. That is when you become thankful for BSAVED Forth. I learned this the very hard way.

The sorting is done only on the name field, but when REV (see later) switches the two file entries, it switches the phone numbers as well as the names. At first I had REV only switching the names, CMOVEing 20 bytes, because I had ALLOTTed only 20 bytes to TEMP and to TEST. When I found that I was sorting the names quite well but was not yet moving the numbers with the names, I just increased the 20 to 35 for CMOVE. That seemed like it worked fine, but after the sorted file was displayed Forth had a SF? ; no amount of key pressing could get Forth off dead center. It did not recognize QUIT, ABORT, MON or any other word that I could find. Power reset was the only answer.

Since the sorting was done completely and correctly I was at loss as to where to look. I worked backwards through Screen #53 verifying each loop by itself and they all worked fine. I thought surely it had something to do with the sorting loops so I did them all single step and they worked by themselves. I finally traced it back to REV which had been working so nicely before. I partitioned it all off but the first 35 CMOVE to TEMP, still ok, Bang!!! It suddenly dawned on me I that I had not increased the ALLOT for TEMP when I increased the CMOVE from 20 to 35. Let that be a lesson to ya!! When you realize that I had to reboot after most every test try, you see that the troubleshooting session got rather frustrating. I get smart slowly.

Located elsewhere herein, is a table showing how the word SORT$ works in detail. If you study this and understand it, you will also understand CSTR$, you may then start wearing your ADVANCED BEGINNER IN FORTH Badge.

When SORT$ is used in the word TST (TeST), Screen #53, a test is made only to see if the flag left by SORT$ is greater than zero, meaning the string at adr1 is greater than the string at adr2. If so the word REV is executed and the strings are exchanged and FLAG is set to 1. TST is used in a loop named SF1 (SortFile 1) which is in turn used in another loop called SF (SortFile). The successive application of TST and REV eventually sorts the strings into alphabetical order. When a test of FLAG by SF shows that there have been no string REVersals in the last pass through SF1, the loop is exited and the words PF (PrintFile) prints the sorted file and .END (PrintEnd) notifies you that it is sorted. The sorting time is only a couple of winks.

Screen #54 is not really part of the essentials, and you may skip it if you like but you will miss all the fun. It is set up to allow you to watch the "bubble sort" process. Actually, when you are watching it looks more like a "settling" process with the heavy strings, like ZIGGY, sinking to the bottom. The words DS (DisplaySort), DSL (DisplaySortLoop), and T (Test) have been embellished from that on Screen #53 to label what is going on and delays have been inserted to slow it down enough to follow.

Please do not procrastinate. Enter these screens and watch how SORT$ works, it is certainly an educational process.

SORT$ And How It Works

The definition of the sorting word SORT$ is,

```
: SORT$ ( adr1 adr2 cnt--) 2DUP + ROT  DO  DROP  1+
DUP  1- C@ I C@ - DUP IF DUP ABS / LEAVE ENDIF LOOP SWAP
DROP ;
```

Assuming that address #1 (adr1), address #2 (adr2) and the count of characters to compare (u), are on the stack at the time of execution of SORT$, the Table below shows what will be on the stack following the execution of each component word of SORT$.

TABLE 1
SORT$ ANALYSIS

WORD	STACK CONTENTS
SORT$	adr1 adr2 u
2DUP	adr1 adr2 u adr2 u
+	adr1 adr2 u (adr2+u)
ROT	adr1 u (adr2+u) adr2
DO	adr1 u (or f later)

[adr2+u and adr2 become the loop limit and starting index value]

DROP	adr1
1+	adr1+1 (set = Z)
DUP	Z Z
1-	Z adr1
C	Z b1 [b1=ch at adr1]
I	Z b1 adr2 [note 1]
C	Z b1 b2 [see b1]
-	Z f [note 2]
DUP	Z f f
IF	Z f

[If b1<>b2 is true, execution goes to the word following IF else, to the word following THEN]

DUP	Z f f
ABS	Z f ABS(f) [note 3]
/	Z (f/ABS(f))[note 4]
LEAVE	[loop limit set = I]
THEN	[marks end of IF seq]

[If b1<>b2 this is the result]	
LOOP	Z (+/-1) [leave loop]
SWAP	(+/-1) Z
DROP	(+/-1) [true flag]

continued on page 17

Newsletter Update

by Bob Relyea

TIBUG (Brisbane), December, 1991: Editorial; The Disk Library; What's New (including a reminder of our faire); Module Library; Basic Computers; Bits & Pieces by Col Christensen; In the P.O. Box; Tips No. 36; Program to type in; The Brother M-1209 Computer; Why Did't Indent (.IN) Work? by Phil Nordstrand and John Owen; Appreciate Your Programmers by Jim Peterson.

February, 1992: Editorial; Trading Post; Bits & Pieces by Col Christensen; What's New; Disk Library; Program to type in; TI Pei advertisement; Basic Computers by Garry Christensen; Funnelweb Tip; Tips No. 37; in The P.O. Box; Sorting, Part 2 (TIsHUG reprint); Air Taxi Review by Jim Peterson; Organize; Basic, Using String Functions by Regena; Working with Numbers by Joe Nollan.

March, 1992: Editorial; In The P.O. Box; Adventure Compendium; Trading Post; Magazine Library; What's New; Fred's Thingamee Parts and Multiplan; Letter to the Editor; Tips No. 38; Basic Computers, The Inside World; Disk Library; Sorting, Part 3 (TIsHUG reprint); Simple Programs by Jim Peterson; Shop Software.

ATICC (Adelaide), February, 1992: Coordinator's Report; Editorial; Puzzler; Problems Reading DV80 Files; The Witching Hour; Programming Tips by Mark Schafer; Double Column Text Formatter; Ontplopper!; Kids Stuff; XB to TI-Artist by Terry Atkinson; XB Tips Number 10 by Jim Swedlow; The "Clock"; Dips Tips by David Perkovic; TI Oddities - The Ultimate.

SPIRIT OF 99, December, 1991: Announcements; Minutes; More on Gemini 10X; Putting It All Together by Jim Peterson; Schedule; The Tigercub Reformatter+; TI Base Tutorial No 23.1.1&2&3 by Martin Smoley; TI World News; Vapourware, Slow ware and no ware at all!

January, 1992: An Index of 1991 articles (similar to the one we do); meeting dates for 1992

February, 1992: Announcements; Conni Clearinghouse; Conni Minutes; From the Mailbox; I Wish; Program of the Month; Schedule; Sliding Block Puzzles; TI-Base Tutorial; TI Casino, a review; Tigercub PrintAll V.1.6; Tips No. 67; TI World News; TI-Writer, Just for us beginners by Jim Lesher.

March, 1992: Announcements; A Quieter PE Box; Conni Calendar; Conni Clearinghouse; From the President; Program Listing; Programming Music the Easy Way; Ramcharged Computers; Saving Paper While Using TI Writer; Some Recent Products Reviewed; TI Base Tutorials; TI World News; What a Shock.

OTTAWA, December, 1991: Two Cents Worth; OTIUG Club Meetings; Fast Extended Basic by Lucie Dorais; I Like Brain Games by Jim Peterson; Hotline Numbers.

January, 1992: Editor's Notes; Two Cents Worth by Bill Gard; New Stuff Demonstrated in Chicago; Fast Extended Basic by Lucie Dorais; The Home Computer by Jim Peterson; Rambles by Stephen Shaw; Hotline Numbers.

February, 1992: Coming Events; Two Cents Worth from the President; Questions and Answers by Jacques Groslouis; Fast Extended Basic (Game of Onecheck) by Lucie Dorais; Installing a 16 Bit Access on a Horizon Ramdisk Under 1M (for Geneve Users(by David Caron; Hints, Tips & Answers, TI Writer or equivalent; Hotline Numbers.

March, 1992: Coming Events; Editor's Notes; The President's Two Cents Worth by Bill Gard; Fast Extended Basic by Lucie Dorais; Programming Music The Easy Way, Part One by Jim Peterson; Hotline Numbers.

LA99ers TopIcs, December, 1991: Ramblin' Thoughts from the President; XBasic Miscellany by Earl Raguse; Crackerbarrel by Chick De Marti; Procedures to Printer by Fred Moore; Mini-Database by Chick De Marti; Announcement of a Convention in Phoenix, Arizona; Disk Drives by Jim Ness; Club Meetings.

January, 1992: XB Miscellany by Earl Raguse; Art Printshop; Editor's Column; Using Transliterates.

February, 1992: President's Message; XB Miscellany by Earl Raguse; Programmer's Dilemma by Don Lester; TI-Base tutorials V 25.1.1&2 by Martin Smoley; How To Do A Person To Person Download by Jeff Overton; TI Bits n' Bytes by Val Mehlima; Archiving- A Headache? by Andy Frueh.

March, 1992: TI-Base Tips by Bill Gaskill; XB Miscellany by Earl Raguse; Tips Index by Ray Frances; Managing Your Money - 4 by Bill Gaskill.

UGOC ROM, December, 1991: President's Message; The Editor Pops Off; The Member Ship; Joke of the Month; November Board Minutes; Hall of Fame; Biography of a Reborn TI/99er; Quotes From the Past; XB Miscellany by Earl Raguse; Dips Chips and Sips; UGOC Interest Survey; Fest West '92.

January, 1992: Editor Pops Off Again; President's Message; Buy/Sell Opportunities; The Member Ship; Untitled Column; Hole Currents by Earl Raguse; We Get Letters; XB Miscellany by Earl Raguse; Tips To Remember by Gene Bohot; EDP Job Description.

February, 1992: Editor Popping Off by Earl Raguse; President's Message; Fibbed To Best My Friend; January Board Minutes; The Member Ship; Salvage Operations; Half a Label by George Clark; Hole Currents No. 2 by Earl Raguse; New Member Biography; XB Miscellany No. 10 by Earl Raguse; Minilabel program; Robot Sound by Earl Raguse; TI goes to College; Dips, Chips and Sips; Fest West.

March, 1992: Editor Popping Off; President's Message; February Board Minutes; The Member Ship; Classic Computer Club; Border Paper Corrections; Quips and Quickies; Letter from Boise; XB Miscellany No. 11 by Earl Raguse; Hole Currents No. 3 by Earl Raguse; Dips, Chips and Sips; More Definitions; Why I Love my TI 99/4A by Earl Raguse; Pastel Colors.

April, 1992: Editor Popping Off; President's Message; March Board Minutes; The Member Ship; Pain In The Neck; Potpouri Page; XB Miscellany No. 12 by Earl Raguse; Hole Currents No. 4 by Earl Raguse; Spaced Out by John Bolda; TIW/Fnlweb Ctrl/Fctn Keys by Newt Armstrong; Impressions Fest West; Dips, Chips and Sips by Siles Bazerman; Evolution of a Computer.

TI FOCUS, December, 1991: The final issue (sad!) of the magazine which contains a lot of thanks and ramblin' thoughts, a reprint of their first newsletter of January, 1983 and a tutorial on TI-Base. They plan of continuing their meetings, though, in 1992.

THE PUG PERIPHERAL, December, 1991: Club News by Gary Taylor; Kiddie Corner; Computerized VCR; Library News; Articulations; How To Make A Disk Case; Trivia; Tips NO. 59.

January, 1992: Club News; From the Librarian; Graphics Comparison; Meeting Minutes; Tips No. 60

THE BOSTON COMPUTER SOCIETY, November, 1991: Listen by Justin Dowling; Is it Time Yet? by Steve Richardson; NEWS from the Editor by Chris Pratt; To Print or Not by Marshall Ellis; Terminology At The Heartbeat by Chip Chapin.

December/January, 1991/2: Listen by Justin Dowling; Barry's Corner by Barry Traver; Programming Tips; From the Teacher's Desk; Hardware Tips (repairing Extended Basic cartridges); Comparing Health Insurance.

February, 1992: Hear Ye! by Justin Dowling; Person to Person Downloads; Fruek Ware; .INdent; Pop Cart; TIdbits No. 54 by Steve Nickelson.

TIdbits (Memphis), October, 1991: President's Bit by Gary Cox; In the News; Government Information; Modem Use, Part 4; Still In Use; From The Teacher's Desk Video Tapes of Lima; Extended Basic; Computers, Vintage; The Next Generation; Remarks on Remarks; Editor's Bit; Program Bit.

December, 1991: President's Bit by Gary Cox; In The News by Gary Cox; Review of Sound F/X; Christmas Eve; My Memory Banks; From the Teacher's Desk; Reformatting by Jim Peterson; Program Bit.

January, 1992: President's Bit; In The News; TI-99/4A Computer Repair; More Video Tapes From Lima; My Memory Banks; From the Teacher's Desk; The Programmers Notebook; Calendars by Charles; Program Bit.

continued on page 23

Regional Group Reports

Meeting Summary For JUNE

Banana Coast	14/06/92 Sawtell
Central Coast	13/06/92 Saratoga
Glebe	11/06/92 Glebe
Hunter Valley	13/06/92?Boolaroo
Illawarra	15/06/92 Keiraville
Liverpool	12/06/92
Northern Suburbs	25/06/92
Sutherland	19/06/92 Jannali

BANANA COAST Regional Group
(Coffs Harbour Environs)

We never miss meeting at Kerry Harrison's residence 15 Scarba St. Coffs Harbour, 2 pm second Sunday of the month. Visitors are most welcome. Contact Kerry 52 3736, Kevin 53 2649, Rex 51 2485 or John 54 1451.

CENTRAL COAST Regional Group

Regular meetings are normally held on the second Saturday of each month, 6.30pm at the home of John Goulton, 34 Mimosa Ave., Saratoga, (043) 69 3990. Contact Russell Welham (043)92 4000.

GLEBE Regional Group

Regular meetings are normally on the Thursday evening following the first Saturday of the month, at 8pm at 43 Boyce Street, Glebe. Contact Mike Slattery, (02) 692 8162.

HUNTER VALLEY Regional Group

The meetings are held regularily at the Boolaroo Ambulance Station. All welcome. Please contact Geoff Phillips on (049) 428 176 for details.

ILLAWARRA Regional Group

Regular meetings are normally held on the second Monday of each month after the TIsHUG Sydney meeting, except January, at 7.30pm, at the home of Geoff & Heather Trott, 20 Robsons Road, Keiraville. A variety of activities accompany our meetings, including Word Processing, Spreadsheets and hardware repairs. At the May meeting Geoff explained how he fixed Bruce's system. Contact Lou Amadio on (042) 28 4906 for more information.

LIVERPOOL Regional Group

Regular meeting date is the Friday following the TIsHUG Sydney meeting at 7.30 pm. Contact Larry Saunders (02) 6447377 (home) or (02) 7598441 (work) for more information.

NORTHERN SUBURBS Regional Group

Regular meetings are held on the fourth Thursday of the month. If you want any information please ring Dennis Norman on (02)452 3920, or Dick Warburton on (02) 918 8132. Come and join in our fun.

Dick Warburton.

SUTHERLAND REGIONAL REPORT

Regular meetings are held on the third Friday of each month at the home of Peter Young, 51 Jannali Avenue, Jannali at 7.30pm.

Peter young
Regional Co-ordinator

TIsHUG in Sydney

Monthly meetings start promptly at 2pm (except for full day tutorials) on the first Saturday of the month that is not part of a long weekend. They are held at the RYDE INFANTS SCHOOL, Tucker Street (Post Office end), Ryde. Regular items include news from the directors, the publications library, the shop, and demonstrations of monthly software.

JUNE MEETING - 6TH JUNE

Among other things, there will be a Special General Meeting to consider changes to our Memorandum of Association. To this end, bring along the May issue of the TND and read the relevant section in the Secretary's report again to familiarise yourself with the contents. Any new software and hardware that come to hand will be demonstrated.

****

The cut-off dates for submitting articles to the Editor for the TND via the BBS or otherwise are:

July	14 June
August	12 July

These dates are all Sundays and there is no guarantee that they will make the magazine unless they are uploaded by 6:00pm, at the latest. Longer articles should be to hand well before the aboves dates to ensure there is time to edit them.

****

TIsHUG Meetings for Sydney

June

A Special General Meeting to consider changes to our Memorandum of Association. New software and hardware to be demonstrated. Watch this space for more details.

July

The second buy, swap and sell day. This one is on the first Saturday of the school holidays but plan to take the day off and see what is about.

August

New software and hardware to be demonstrated. Watch this space for more details.

September

The second all day tutorial session. Your chance to learn about using software, writing programs or understanding hardware. We can provide anything that you want but you must tell us what you would like and at what level you would like it.

October

The third buy, swap and sell day. This one is in the middle of the school holidays but plan to take the day off and see what is about.

November

The TI-Faire will be a few weeks after this meeting so it may be taken up with the organizational requirements of this big day. New software and hardware to be demonstrated. Watch this space for more details. Time to think about nominating for positions on the board. I am sure there will be some vacancies this year!

December

The Annual General Meeting followed by some festive eats and drinks, There will probably be a bit of celebration after the TI-Faire, if we are all still friendly after the event. Make sure that you attend and give your support to all the workers in the club.

continued from page 22

February, 1992: President's Bit; In The News; For Sale; A Ghost From The Past; President's Day Notes; Compare Health Insurance; Subprogram Falsehoods; Extended Basic, Good Choice; Catalogue; Suppliers For TI.

March, 1992: President's Bit; In The News; From The Teacher's Desk; Printer Control Codes; TI-Base Tips by Bill Gaskill; To Charge or Not to Charge (TIsHUG reprint); New Product News?; I Got It Back; Word Fun, Part 1; Programmers Dilemma; Program Bit.