Volume 1 Issue 5

C Vu

The Journal Of The C Users' Group (U.K.)

Mail: 36 Whetstone Close, Farquhar Road, Birmingham, B15 2QN
Telephone: 021-454-3448

Editorial

Hi! Welcome to Issue 5 of C Vu, the Journal of the C Users' Group (U.K.). This is also the fifth, and last, issue I shall be editing and maling up myself – for a while at least. Unless you're in the business yourself, you'll be amazed at the amount of time it actually takes to get a half decent layout! Anyway, as from Issue 6, Martin Houston will be taking over the helm. If you would like to take it over yourself, I'm sure Martin is open to offers! We have been let down once already, but if you are sure you can spare the time and, obviously, have the equipment (I can flog you a second-hand copy of Timeworks Desktop Publisher if you have an Atari ST) then give Martin a ring on 021-454-3448.

Right onto the interesting things. Another good packed issue this time around – hopefully something for everyone. If there isn't – then you know the cure – either write it yourself, or write to us telling us what you're looking for from C Vu (we aren't telepathic!).

I know we have already had quite a few new members since the last issue, and te PC(W) Show in September should see another surge, so I'll give a quick resume of the what the group is about. At present the group has three main activities: the one you're looking at! (which we currently publish quarterly, although if we get sufficient material, it could become more frequent), the Source Library – a collection of public domain C source code available at minimal cost to all members (more details on that inside), and finally local groups. If you are interested in either setting up a local, or special interest group then read Francis Glassborow's column in this issue. When you've read it – do something about it!

We also want to hear from each and every one of you – telling us what you're doing, what projects you're working on, what you think of the group, what you like and dislike about C, any bugs or problems you've encountered along the way. We'll do all we can to help you get the most out of C, and the C Users' Group (U.K.) - it is, after all, your group! We are not a commercial, profit making group. What we aim to be is professional – with your input.

Phil Stubbington
Editor

Contents

Wotz Nu?
by Phil Stubbington

More things to spend your, or preferably someone else's, money on!

The South West Software Library

Definitely the cheapest thing to tell you about this month is the latest catalogue (Number 3) from the South West Software Library. Dealing exclusively in public domain/shareware software for the Atari ST, the S W S L are the only professional P.D.library I know of.

The twenty page catalogue contains numerous hints and tips for getting the software to work on your system. In addition, the library will do as much as humanly possible to solve any compatibility problems for you (no guarantees though!). They also offer upgrades - either for free (if you order other disks at the same time) or for a nominal handling charge.

Modesty forbids me from mentioning the fact that two of my programs are available in the catalogue (one on the front cover!). S.W.S.L. have quite a good selection of stuff for the Atari ST C programmer - I counted 28 disks all with a very high content of C source code and/or C programming utilities.

All standard volumes are £3:00 each, or £2:50 if you order 4 or more at a time. For a copy of the catalogue, just write to: The South West Software Library, PO Box 562, Wimborne, Dorset, BH21 2YD

Watcom C, Version 7

If you are getting a bit bored with M*cr*s**t C, the latest version of the Canadian Watcom compiler looks set to blow it right out of the water. Watcom stands a very good chance of being the first ANSI validated compiler, but in the mean time offers better optimisation than MS C (smaller executable and more dhrystones). Watcom comes in at a very respectable £250 (about the same price as M*cr*s**t C!), or if your budget doesn't stretch that far, Watcom Express is a mere £60. A full review will appear soon - if you can't wait contact the ubiquitous

Grey Matter on: 0364-53499 or at: 4 Prigg Meadow, Ashburton, Devon, TQ13 7DF

Swifte-C For Mirage

Sahara Software Ltd are due to start shipping the "Swifte-C Programming Language Development System" for MIRAGE this month (see issue 3 for a review of Mirage). Mirage itself is a multi-user, multi-tasking OS available for the Motorola 680x0 family and has been implemented on a wide range of systems (including the Atari ST and Commodore Amiga). Prior to this announcement, the only C compiler available under Mirage was the singularly unexciting Lattice C V3.03. In addition to C, a wide variety of languages including Fortran, Pascal, Lisp, BASIC and Assembler are available. Swifte-C looks set to live up the extremely high standard set by previous in-house developments.

Feature wise: the compiler is fully K&R compatible, and Swifte have pledged support for the ANSI standard. In keeping with their previous language products Swifte-C, and subsequent code, is said to be compact, fast and re-entrant. Swifte-C supports the 68020 and 68881, and no limitations are

imposed on code or data size.

For further details contact: Francis Cox on 01-922-8850, or at: Sahara Software Ltd, South Bank Technopark, 90 London Road, London SE1 6LN

Turbo C For The Atari ST?

Although rumoured for some time, the appearance of Turbo C must rate as one of the most unexpected products of all time! Details are still quite sketchy, but the compiler is said to be fast and well optimised. Floating point routines are nothing to shout about, and the manual and on-line help are in German (an English translation will probably be available by the time you read this). Rumour has it that Calamus, the heavy weight DTP package, was written using Turbo C, so it is obviously quite a solid product (and who needs floating point anyway?). For further information don't bother contacting Borland - they'll just refer you to the guys and girls state-side (who probably know nothing about it either!).

Instead, drop a line to: Softpaquet, PB6250, 2702 AG Zoetermeer, Netherlands

Microsoft C Version 5.1 from Steven W. Palmer, Membership # 8712

Now optimizing and capable of handling OS/2 and DOS. Steven Palmer puts the latest version under the microscope.

Microsoft's latest C compiler comes courtesy of a huge package seating three voluminous manuals, a quick reference guide, a Quick-C Programmers Guide and at least eleven disks. For people used to dealing with the company, this should come as no surprise. The whole package exudes an air of professionalism.

Versions 5.1 and later caters for OS/2 as well as DOS development. OS/2 executable versions of the compiler, editor, debugger and dynamic link libraries are supplied together with a number of batch files which let you create bound versions for running in both environment. We only have MS-DOS at the moment, so I can't comment on the OS/2 side of the package. This whole review is based on the DOS version.

Two of the 5.25" disks are in high-density AT format and contain all OS/2 specific files. The other disks can be read by any standard XT disk drive. You can order disks in either 5.25" or 3.5" format, and Microsoft provide a free conversion from one format to another. We just linked an XT and a PS/2 and used Procomm to copy them over (it took 6 hours to do the lot!)

The Manuals

The four manuals supplied include: "User's Guide and Mixed-Language Programming Guide", "Language Reference, Microsoft CodeView, Microsoft Editor and Utilities" guide, a Run-Time Library Reference and the "Quick-C Programmer's Guide". The quick reference guide is thicker than the one supplied with version 4.0, and lists all command line options for the compiler, debugger and other utilities, and a brief synopsis of all library functions. It is ring bound, and has a stiff cover so it could easily be persuaded to stand up next to your computer.

The Quick-C manual is a bound paperback, 400 pages long. Most of the manual leans more towards the beginner to C than the other three and contains chapters introducing the C language. These

chapters, however, are very brief and are more suitable for an experienced programmer coming from another language than for someone just learning to program. Other chapters describe the graphics library, the Quick-C environment and the other tools on the disks, however no run-time library guide is included.

The other manuals are much the same as those supplied with version 5.0, except that the binders have been made larger to accommodate the extra information. If you paid for the cheap upgrade from version 5.0, then you won't get the new binders - you'll have to squeeze the extra pages into your old ones. Whether you pay the full price or upgrade, the contents of the manuals are exactly the same.

The run-time library manual is well organised with at most one function per page, together with prototypes, descriptions, examples and cross-references. An appendix at the back describes differences between routines and variables common to MS-DOS and Xenix. It is easily the best documented run-time library that I have seen, and it is very extensive.

The utilities manual contains a user guide for the CodeView source code debugger, the Microsoft editor, the linker, librarian and other small utilities. The rest of the manual contains a C language reference guide which makes concise reading. Nothing in the package makes any attempt to teach you C programming - Microsoft assumes that you have access to other literature and even lists some recommended sources.

The last manual is thinner and describes the C compiler in considerable detail. There are chapters on writing portable programs, programs that will eventually get blown into ROM and discusses memory models and cross-language programming. Very little is omitted and it is well organised. In the six months that I have had the package, I have hardly found an occasion where I couldn't access a certain piece of information.

All three manuals are indexed and have subject divider tabs for quick access to the required section. They also make ideal weight training during those slack lunch hours.

Installation

Microsoft now supply a SETUP.EXE product to automate the installation of the compiler and other tools. This is standard with almost all their other products and is certainly easier than the D-I-Y approach offered in version 4.0. Running the setup program offers you a list of possible configurations: DOS or OS/2 versions, or a bound copy that will run in both modes; small, medium, compact or large model libraries; floating point emulation or coprocessor support; different editor configurations emulating Brief(tm), WordStar or the Epsilon editor and many other minor options. If you elect to install the whole lot, you will need at least 8Mb spare hard disk space!

The MSC.EXE compiler driver is no longer supplied with the package, and has been replaced by the command line driven CL.EXE utility for Xenix compatibility. If you have any batch files containing MSC, then Microsoft offer guidance on converting them to the CL format.

Library

There are separate run-time libraries for each memory model, and during installation, you are given the option to combine the floating points library and graphics library together with the main library. Such combined libraries take up more disk space, but linking is faster as a result. The total size of the combined libraries for all four memory models is just over 1Mb.

The major improvements over version 4.0 is the inclusion of bit-mapped graphics functions and low-level DOS and BIOS functions. The graphics functions support all graphics modes from CGA to VGA, and even Hercules monochrome graphics. However compared with most commercial graphics libraries, Microsoft's offering is slightly limited. Text can only be output at text screen coordinates, and no text styles are supported.

CodeView

The CodeView debugger is much larger than it's earlier incarnations (it weighs in at over 200k, and takes minutes to load from floppy!) as it now tries to cater, not only for C programmers, but for BASIC, Pascal, FORTRAN and assembler programmers too! The expression evaluator is different for each mode, hence the increased size. It now handles 80386 code and lets you debug overlays. It is still my favourite debugger and a far cry from those old days on the Harris minicomputer when I had to "printf()" every function to track down the bugs!

In order to debug your source code with CodeView, you need to add a compiler switch when you invoke the compiler. If you link your program separately, you will also need to tell the linker to add CodeView debugging information onto the end of your executable file. As a consequence, such files are much larger than normal. If your program is very large, then CodeView may not be able to load the whole program, even if MS-DOS can. To get round this, Microsoft now include a CVPACK utility which optimises the debugging information in the executable file. On one test, 32654 bytes of debugging info was packed to 3856 byte just by running CVPACK.

An enhancement to the old CodeView is the facility to view structures. The debugger recognises nested structures, so moving the cursor to the name of a nested structure and pressing the RETURN key opens up another window showing the contents of that structure. You can use this feature to walk through a linked list without needing to deal with memory dumps. The same command works for structures in BASIC, Pascal and MASM.

The protected mode version of CodeView runs under OS/2 and allows debugging of separate threads. You can run threads at full speed, switch between threads or freeze background threads while others run at their normal speed.

Microsoft Editor

The new Microsoft Editor is something of a disappointment in comparision. It is a fully programmable editor which allows users to extend it's facilities by writing extension programs in C and loading them when the editor is started. However the editor is fairly awkward to use in comparision with Brief, my standard editor, and hints somewhat at an EMACS ancestry. The editor insists on writing modified files back to disk each time that you switch buffers, and there is no way to exit the editor without having it write back out all files which have been modified. You can set an internal switch to change this behaviour, but details on how to do this is buried somewhere in an obscure place at the back of the manual.

The editor can be configured to emulate other editors. Files to emulate Brief, Epsilon and the WordStar/Quick-C keyboards are already supplied. However when I tried the Brief configuration, I found several common keystrokes were unemulated or different. I couldn't move to the end of the file using Ctrl and End - the editor reported that the keystrokes were unassigned! However I was able to modify the configuration file to make it closer to Brief. Even so, I finally gave up and went back to Brief.

The Compiler

The compiler consists of four programs, CL.EXE, C1.EXE, C2.EXE and C3.EXE which contain the main program, the preprocessor and lexical analyser, the parser and the code generator and optimiser respectively. For large programs, a large model version of C1.EXE is supplied which can compile programs that normally cause heap errors under the default version. The large model version of C3.EXE is no longer supplied.

Compilation is noticably faster than earlier versions and generates better quality code. The optimiser now handles loop optimisation and common subexpression elimination in addition to those available in version 4. Some library functions can be compiled inline, which improves overall program speed by eliminating the overhead of calling them as extrinsic functions. However it still falls down on some occasions, such as failing to eliminate redundant storage allocations. Be warned that some programs compiled on full optimisation will probably behave differently from the non-optimised versions. To be fair, Microsoft recognise this and explain in a README.DOC file how this can be overcome.

The compiler is closer to the proposed ANSI standard than previous incarnations but some features are still unsupported. The 'volatile' declarator is implemented syntactically, but not semantically. Trigraphs are not included, but then they would be redundant as the IBM environment supports the full 8-bit ASCII character set. However most other features are fully supported, including the use of '//' as single line comment delimiter.

New #pragmas allow changing the source and object code listing format and placing comments into the object file. While testing this, I found a bug. The string

```
#pragma comment(lib, mylibry)
```

mentioned in the manual will not compile. If it had worked, the 'lib' comment type would allow the user to select, from the source code, a library to be included at link time.

Other features include #error to output an error message, new predefined macro names __DATE__, __TIME__, __STDC__ and __TIMESTAMP__, the use of macros in #include and #line directives and ellipsis in function prototypes to specify that the function takes a variable number of arguments. All these features can be disabled via a command line option if portability to non-ANSI compilers is an issue.

Quick-C

As a result of the recent publicity, most people have probably heard of the Microsoft Quick-C development environment. This integrates a full screen editor with a fast in-memory C compiler, debugger, linker and make utility. The latest version (2.0) has a good hypertext-like help system but the one supplied with our compiler (version 1.03) isn't all that bad. Highlighting a function name in your source code and selecting help brings up a window that shows the function prototype and the name of the #include file in which it is defined.

The product is the largest single program supplied in the package. It is over 300k in size (326k to be exact) and requires at least 512k memory to run in. It doesn't use expanded memory, so be prepared to have to shove out a few TSRs to get it to fit in. And another thing - don't even consider running it from a dual floppy machine. It takes practically a lifetime to load.

The editor emulates WordStar up to a point. The user manual doesn't list all the keystrokes, and the on-line help only lists a few. You can have two files in memory at once, but when switching between them, the program prompts whether you want to save the active file to disk. You can

disable the prompt but not the writing to disk.

I used Quick-C to develop a project comprising several C source code files, but I soon had to switch back to Brief and the command line compiler when I began running out of memory. Quick-C is good for quick prototyping or small programs, but anything larger just won't fit. Furthermore, assembler modules are not handled, and not all library functions are available in Quick-C. If you need the missing functions, you have to make your own Quick-library and load that each time you start Quick-C. The process is quite laborious and not really worth the effort.

Compilation is really fast, and options are available to write the object code to memory, an .OBJ file or an .EXE file. Minimal optimisation is supported, and all files are compiled using the medium memory model (single data segment, multiple code segments) - there is no support for any other memory models. If errors are detected, the last 26 errors are remembered and displayed at the bottom of the screen while you make the corrections. Linker errors, however, get written to

a separate file and you need to load and read this file to fix them.

I discovered that Quick-C doesn't always generate the same object code as it's big brother. Some complex macros cause it to throw up, and others get compiled wrong. So if you find a really tricky bug, have a go at compiling the offending code with the optimizing compiler and seeing if the code works then. This is an approach that I've had to take a few times. As far as Quick-C is concerned, the moral seems to be:keep it small and simple. The README.QC file supplied with the package lists all potential problems.

The debugger can be called up anytime, and only works on source lines. The user interface is a subset of CodeView, but there is no provision for viewing the object code. You cannot view the contents of variables without placing them as watchpoints, and there is no means of examining any memory location unless a pointer happens to be pointing to it. As a serious development system, Quick-C is very limited, though I understand that the latest version corrects most of these deficiencies.

The Cost

If you already own version 5.0, you can upgrade cheaply to the latest version through Microsoft UK. If you purchased version 5.0 before 1st March 1988, the cost is £57.50 (including VAT). If you purchased after that date, then the upgrade is free! Owners of earlier versions of the compiler can upgrade for only £109.50 (including VAT). Otherwise you pay £275.00 for the full package. When upgrading, you will need to return your old master diskettes as proof of purchase.

For upgrades, contact Microsoft UK at Excel House, 49 De Montfort Road, Reading, Berkshire, RG1 8LP, telephone (0734) 500741, telex 847924 Microe G, facsimile (0734) 507624.

Conclusions

The product is not without its weak points. For me, these are the Quick-C environment and the Microsoft Editor, but I suspect that other people may find these products more useful than I did. The compiler itself is fast, generates very compact code and provides a whole range of features and options. For the cost, it represents a truly professional package and is unreservedly recommended.

Writing Portable C Programs
by Ron Wellsted

Part II

The Run-Time Environment

Since the C language has been implemented on many (if not most) different types of computer, from 8-bit microcomputers up to super-computers such as the Cray family, so a program (particularly public domain programs) may end up running on a totally different machine to the one for which it was originally written.

Data Types

While sizeof() will return the number of bytes in a particular data type, care should be taken to avoid assumptions about the actual size of a particular type. Normally, a char occupies 1 byte, a short 2 bytes and a long 4 bytes. These are the only assumptions that can be made with any degree of safety. An int is not included in this list because the size varies, on an Intel 8086 based system, it occupies 2 bytes, whereas on a Motorola 68000 based system, it occupies 4 bytes (To confuse matters further, an int is 4 bytes on an Intel 80386 in native mode). If binary data interchange is required between systems, attention must be paid to structure packing (mentioned in part 1) and to byte and word ordering.

CPU	Short a0 a1	Long a0 a1 a2 a3
PDP-11	b0 b1	b2 b3 b0 b1
VAX-11	b0 b1	b0 b1 b2 b3
i80x86	b0 b1	b0 b1 b2 b3
680x0	b1 b0	b3 b2 b1 b0

Byte ordering for short and long types

UNIX uses PDP-11 ordering as the reference point, so VAXes and Intel chips are WORDSWAPPED, while a 680x0 is BYTESWAPPED

Operating Systems

A program will normally use the operating system for file operations and process control. In general, the operating system of a computer is hidden behind the run-time library, but differences will always occur, mostly in the file naming conventions. For example, a file could have a full path-name under UNIX of /usr/ronw/lib/filename, for MSDOS it could be c:\usr\ronw\lib\filename or c:/usr/ronw/lib/filename (in any mix of upper or lower case!), while under VMS it could be SYS $SYSTEM:[USR RONW LIB]FILE NAME.

High Level I/O

The high level functions (fopen, fclose, fread, fwrite, fprintf, etc) are the most portable of the file i/o functions. The only differences normally occur on the modes for fopen, but ``r'' (read) ``w'' (write) and ``a'' (append) should work without problems.

Low Level I/O

The standard low level functions (open, close, read, write & lseek) are direct usually direct implementations of the UNIX system calls to perform these functions. The only problems usually occur with the open function. To avoid problems, do not use numbers for the open mode or create mode arguments, instead use the #defined constants

Text or Binary Mode?

This problem DOES NOT occur under UNIX, but does occur under MSDOS and some other operating systems. The problem is due to the method the operating system uses to mark a new-line (`\n') in files UNIX uses the ASCII line feed character, while MSDOS us es a carriage return/line feed pair. In text mode a cr/lf pair will be translated to a single lf when reading from a file and a lf will be translated to a cr/lf pair when writing a file.

This has a unpleasant side-effect of the file having a different size to the number of bytes read or written! So never use stat() or fstat() to predict the number of bytes to read, but use return value of read() or feof() to detect end of file. Also read() may return less bytes than were asked for when not at the end of file. In binary mode no translation will take place, so that binary data can be read or written, but this may have weird effects if you attempt to read or write text files. Text mode is normally the default mode (although some compilers allow the default to be changed) and there is usually a method to set text or binary mode on a per file basis. The method of setting the mode on a file is normally done by a modifier to the open mode (Microsoft C5.1 uses O_BINARY or O_TEXT for open() and ``b'' or ``t'' for fopen()). For example:

```
fopen(textfile, ``rt'')
```

or

```
open(``file bin'', O_WRONLY | O_BINARY).
```

Text mode also slows down file i/o, so it is best to use binary mode if you want to obtain benchmark timings

Process Control

This area is almost entirely dependent upon the O/S. About the only portable function is the system() call. In general, it is best to avoid process control functions if possible.

Hardware

If a program is to be kept portable, it SHOULD NOT make any assumptions about what hardware is present (this includes screen management).

Console I/O

Ideally, all console I/O should be done with stdin and stdout. (The MIT X/Windows system is fast becoming the standard for graphical interfacing in the UNIX world)

Hardware I/O

For portability, a program should never try to access memory or I/O ports directly since the addresses of I/O ports and system variables can change between different versions of the O/S or the machine type (IBM PC and PS/2 is a good example). However, most compilers provide some sort of port I/O function (inp() and outp() are common). If it is really vital to access a specific memory location then a pointer can always be forced to a specific address (declare a union of a long or short and a char *), but do so entirely at your own risk. Neither the author nor the C Users' Group (U.K.) can be held responsible for any damage!

Interrupts, Traps & Exceptions

The signal() function is about the only portable interrupt/trap/exception handler. SIGINT usually corresponds to a user abort (Ctrl-C or Ctrl-Break), while SIGFPE can be used to catch floating point errors.

Proposed Standard For CUG (U.K.) Library Programs

With the very wide range of machines and operating systems used by CUG members, I would like to the following portability standard for use with programs submitted to the library.

The list is not intended to be exhaustive - your suggestions and comments are welcome! (ED - how about operating environments? Windows/PM, Macintosh, GEM, X)

Operating System Dependencies

```
#ifdef UNIX Generic UNIX
#ifdef ATT AT&T specific UNIX
#ifdef M_XENIX Xenix (auto defined by cc)
#ifdef V7 Version 7 UNIX
#ifdef SYS3 UNIX/XENIX system III
#ifdef SYS5 UNIX/XENIX system V
#ifdef BSD BSD specific UNIX
#ifdef BSD42 BSD Release 4 2
#ifdef BSD43 BSD Release 4 3
#ifdef MSDOS Generic MS/PC-DOS
#ifdef OS2 Generic OS/2
#ifdef OS9 Generic OS-9
#ifdef CPM for diehards
```

Hardware or Machine Dependencies

```
#ifdef AMIGA Commodore AMIGA series
#ifdef ATARIST ATARI ST series
#ifdef CRAY wishful thinking!
#ifdef IBMPC Generic IBM PC & Clones
#ifdef IBMPS2 Generic IBM PS/2 & Clones
#ifdef SUN Generic SUN workstation
#ifdef APOLLO Generic Apollo workstation
```

CPU Type Dependencies

```
#ifdef M_I86 Intel i8086 family
#ifdef M_I186 Intel i80186
#ifdef M_I286 Intel i80286
#ifdef M_I386 Intel i80386 & i80386SX
#ifdef MC68K Motorola 68000 family
#ifdef SPARC Sun Microsystems SPARC
#ifdef Z80 Zilog Z80
```

UNIX is a registered trademark of AT&T XENIX is a registered trademark of Microsoft Atari and Atari ST are trademarks of the Atari Corp OS/2 and PS/2 are registered trademarks of I B M Apologies for the use of any unacknowledged trademarks

Professional GEM
by Tim Oren

Part Two – more on windows!

EXCELSIOR!

In this installment, we continue the exploration of GEM's window manager by finding out how to process the messages received by an application when it has a window defined on the screen.

Also, beginning with this column, sample C code demonstrating the techniques discussed will be available on SIG*ATARI in DL5. This will allow you to download the code without interference by the CIS text-formatter used by ANTIC ONLINE output.

The file for this column is GEMCL2.XMO. All references to non-GEM routines in this column refer to this file. Please note that these files will not contain entire programs. Instead, they consist of small pieces of utility code which you may copy and modify in your own programs.

REDRAWING WINDOWS

One of the most misunderstood parts of GEM is the correct method for drawing within a window. Most requests for redrawing are generated by the GEM system, and arrive as messages (read with evnt_multi) which contain the handle of the window, and the screen rectangle which is "dirty" and needs to be redrawn.

Screen areas may become dirty as a result of windows being closed, sized down, or moved, thus "exposing" an area underneath. The completion of a dialog, or closing of a desk accessory may also free up a screen area which needs to be redrawn. When GEM detects the presence of a dirty rectangle, it checks its list of open windows, and sends the application a redraw message for each of its windows which intersects the dirty area.

CAVEAT EMPTOR

GEM does not "clip" the rectangle which it sends to the application; that is, the rectangle may not lie entirely within the portion of the window which is exposed on the screen. It is the job of the

application to determine in what portion of the rectangle it may safely draw. This is done by examining the "rectangle list" associated with the window.

A rectangle list is maintained by GEM for each active window. It contains the portions of the window's interior which are exposed, i.e., topmost, on the screen and within which the app may draw.

Let's consider an example to make this clear. Suppose an app has opened two windows, and there are no desk accessory windows open. The window which is topmost will always have only one rectangle in its list. If the two are separate on the screen, then the second window will also have one rectangle. If they overlap, then the top window will "break" the rectangle of the bottom one. If the overlap is at a corner, two rectangles will be generated for the bottom window. If the overlap is on a side only, then three rectangles are required to cover the exposed portion of the bottom window. Finally, if the first window is entirely within the second, it requires four rectangles in the list to tile the second window.

Try working out a few rectangle examples with pencil and paper to get the feel of it. You will see that the possible combinations with more than two windows are enormous. This, by the way, is the reason that GEM does not send one message for each rectangle on the list: With multiple windows, the number of messages generated would quickly fill up the application's message queue.

Finally, note that every app MUST use this method, even if it only uses a single window, because there may be desk accessories with their own windows in the system at the same time. If you do not use the rectangle lists, you may overwrite an accessory's window.

INTO THE BITS

First, we should note that the message type for a redraw request is WM_REDRAW, which is stored in msg[0], the first location of the message returned by evnt_multi. The window handle is stored in msg[3]. These locations are the same for all of the message types being discuss. The rectangle which needs to be redrawn is stored in msg[4] through msg[7].

Now let's examine the sample redraw code in more detail. The redraw loop is bracketed with mouse off and mouse on calls. If you forget to do this, the mouse pointer will be over-written if it is within the window and the next movement of the mouse will leave a rectangular blotch on the screen as a piece of the "old" screen is incorrectly restored.

The other necessary step is to set the window update flag. This prevents the menu manager from dropping a menu on top of the screen portion being redrawn. You must release this flag at the end of the redraw, or the you will be unable to use any menus afterwards.

The window rectangles are retrieved using a get-first, get-next scheme which will be familiar if you have used the GEM DOS or PC-DOS wildcard file calls. The end of the rectangle list has been reached when both the width and height returned are zero. Since some part of a window might be off-screen (unless you have clamped its position - see below), the retrieved rectangle is intersected with the desktop's area, and then with the screen area for which a redraw was requested. Now you have the particular area of the screen in which it is legal to draw. Unless there is only one window in your application, you will have to test the handle in the redraw request to figure out what to put in the rectangle.

Depending on the app, you may be drawing an AES object tree, or executing VDI calls, or some combination of the two. In the AES case, the computed rectangle is used to specify the bounds of the objc_draw. For VDI work, the rectangle is used to set the clipping area before executing the

VDI calls.

A SMALL CONFESSION

At the beginning of this discussion, I deliberately omitted one class of redraws: those initiated by the application itself.

In some cases a part of the screen must be redrawn immediately to give feedback to the user following a keystroke, button, or mouse action. In these cases, the application could call do_redraw directly, without waiting for a message.

The only time you can bypass do_redraw, and draw without walking the rectangle list, is when you can be sure that the target window is on top, and that the figure being drawn is entirely contained within it.

In many cases, however, an application initiated redraw happens because of a computed change, for instance, a spreadsheet update, and its timing is not crucial. In this instance, you may wish to have the app send ITSELF a redraw request.

The main advantage of this approach is that the AES is smart enough to see if there is already a redraw request for the same window in the queue, and, if so, to merge the requests by doing a union of their rectangles. In this fashion, the "blinky" appearance of multiple redraws is avoided, without the need to include logic for merging redraws within the program.

A utility routine for sending the "self-redraw" is included in the down-load for this article.

WINDOW CONTROL REQUESTS

An application is notified by the AES, via the message system, when the user manipulates one of the window control points. Remember that you must have specified each control point when the window was created, or will not receive the associated control message.

The most important thing to understand about window control is that the change which the user requested does not take place until the application forwards it to the AES. While this makes for a little extra work, it gives the program a chance to intervene and validate or modify the request to suit.

A second thing to keep in mind is that not all window updates cause a redraw request to be generated for the window, because the AES attempts to save time with raster moves on the screen.

Now let's look at each window control request in detail. The message code for a window move is WM_MOVED. If you are willing to accept any such request, just do:

```
wind_set(wh, WF_CXYWH, msg[4], msg[5], msg[6], msg[7]);
```

(Remember that wh, the window handle, is always in msg[3]).

The AES will not request a redraw of the window following this call, unless the window is being moved from a location which is partially "off-screen". Instead, it will do a "blit" (raster copy) of the window and its contents to the new location without intervention by the app.

There are two constraints which you may often wish to apply to the user's move request. The first is to force the new location to lie entirely within the desktop, rather than partially off-screen. You can do this with the rc_constrain utility by executing:

```
rc_constrain(&full, &msg[4]);
```

before making the wind_set call. (Full is assumed to contain the desktop dimensions.)

The second common constraint is to "snap" the x-dimension location of the new location to a word boundary. This operation will speed up GEM's "blit" because no shifting or masking will need to be done when moving the window. To perform this operation, use align() before the wind_set call:

```
msg[4] = align(msg[4], 16);
```

The message code for a window size request is WM_SIZED. Again, if you are willing to accept any request, you can just "turn it around" with the same wind_set call as given for WM_MOVED.

Actually, GEM enforces a couple of constraints on sizing. First, the window may not be sized off screen. Second, there is a minimum window size which is dependent on the window components specified when it was created. This prevents features like scroll arrows from being squeezed into oblivion.

The most common application constraint on sizing is to snap the size to horizontal words (as above) and/or vertical character lines. In the latter case, the vertical dimension of the output font is used with align().

Also, be aware that the size message which you receive specifies the EXTERNAL dimensions of the window. To assure an "even" size for the INTERNAL dimensions, you must make a wind_calc call to compute them, use align() on the computed values, back out the corresponding external dimensions with the reverse wind_calc, and then make the wind_set call with this set of values. A window resize will only cause a redraw request for the window if the size is being increased in at least one dimension. This is satisfactory for most applications, but if you must "reshuffle" the window after a size-down, you should send yourself a redraw (as described above) after you make the wind_set call. This will guarantee that the display is updated correctly. Also note that the sizing or movement of one window may cause redraw requests to be generated for other windows which are uncovered by the change.

The window full request, with code WM_FULLED, is actually a toggle. If the window is already at its full size (as specified in the wind_create), then this is a request to shrink to its previous size. If the window is currently small, then the request is to grow to full size.

Since the AES records the current, previous, and maximum window size, you can use wind_get calls to determine which situation pertains. The hndl_full utility in the down-load (modified from Doodle), shows how to do this.

The "zoom box" effects when changing size are optional, and can be removed to speed things up. Again, if the window's size is decreasing, no redraw is generated, so you must send yourself one if necessary. You should not have to perform any constraint or "snap" operations here, since (presumably) the full and previous sizes have had these checks applied to them already. The WM_CLOSED message is received when the close box is clicked. What action you perform depends on the application. If you want to remove the window, use wind_close as described in the last column. In many applications, however, the close message may indicate that a file is to be saved, or a directory or editing level is to be closed. In these cases, the message is used to trigger this action before or instead of the wind_close. (Folders on the Desktop are an example of this situation.)

The WM_TOPPED message indicates that the AES wants to bring the indicated window to the "top" and make it active. This happens if the user clicks within a window which is not on top, or if the currently topped window is closed by its application or desk accessory. Normally, the

application should respond to this message with:

```
wind_set(wh, WF_TOP, 0, 0);
```

and allow the process to complete.›› In a few instances, a window may be used in an output only mode, such as a status display, with at least one other window present for input. In this case, a WM_TOPPED message for the status window may be ignored. In all other cases, you must handle the WM_TOPPED message even if your application has only one window: Invocation of a desk accessory could always place another window on top. If you fail to do so, subsequent redraws for your window may not be processed correctly.

WINDOW SLIDER MESSAGES

If you specify all of the slider bar parts for your window, you may receive up to five different message types for each of the two sets of sliders. To simplify things a little, I will discuss everything in terms of the vertical (right hand side) sliders. If you are also using the horizontal sliders, the same techniques will work, just use the alternate mnemonics. The WM_VSLID message indicates that the user has dragged the slider bar within its box, indicating a new relative position within the document. Along with the window handle, this message includes the relative position between 1 and 1000 in msg[4].›› Recall from last column's discussion that this interval corresponds to the "freedom of movement" of the slider. If you want to accept the user's request, just make the call:

```
wind_set(wh, WF_VSLIDE, msg[4], 0, 0, 0);
```

(Corresponding horizontal mnemonics are WM_HSLID and WF_HSLIDE).

Note that this wind_set call will not cause a redraw message to be sent. You must update the display to reflect the new scrolled position, either by executing a redraw directly, or by sending yourself a message.

If the document within the window has some structure, you may not wish to accept all slider positions. Instead you may want to force the scroll position to the nearest text line (for instance). Using terms defined in the last column, you may convert the slider position to "document units" with:

```
top_wind = msg[4] * (total_doc - seen_doc) / 1000 + top_doc
```

(This will probably require 32-bit arithmetic).

After rounding off or otherwise modifying the request, convert it back to slider units and make the WF_VSLIDE request.

The other four slider requests all share one message code: WM_ARROWED. They are distinguished by sub-codes stored in msg[4]: WA_UPPAGE, WA_DNPAGE, WA_UPLINE, and WA_DNLINE. These are produced by clicking above and below the slider, and on the up and down arrows, respectively. (I have no idea why sub-codes were used in this one instance.) The corresponding horizontal slider codes are: WA_LFPAGE, WA_RTPAGE, WA_LFLINE, and WA_RTLINE.

What interpretation you give to these requests will depend on the application. In the most common instance, text documents, the customary method is to change the top of window position (top_wind) by one line for a WA_UPLINE or WA_DNLINE, and by seen_doc (the number of lines in the window) for a WA_UPPAGE or WA_DNPAGE.

After making the change, compute a new slider position, and make the wind_set call as given above. If the document's length is not an even multiple of "lines" or "pages" you will have to be careful that incrementing or decrementing top_wind does not exceed its range of freedom: top_doc to (top_doc + total_doc - seen_doc).

If you have such an odd size document, you will also have to make a decision on whether to violate the line positioning rule so that the slider may be put at its bottom-most position, or to follow the rule but make it impossible to get the slider to the extreme of its range.

A COMMON BUG

It is easy to forget that user clicks are not the only things that affect slider position. If the window size changes as a result of a WM_SIZED or WM_FULLED message, the app must also update its sliders (if they are present). This is a good reason to keep the top of window information in "document units".

You can just redo the position calculation with the new "seen_doc" value, and call wind_set. Also remember that changing the size of the underlying document (adding or deleting a bottom line, for instance) must also cause the sliders to be adjusted.

DEPT. OF DIRTY TRICKS

There are two remaining window calls which are useful to advanced programmers. They require techniques which I have not yet discussed, so you may need to file them for future reference.›› The AES maintains a quarter-screen sized buffer which is used to save the area under alerts and menu drop-downs. It is occasionally useful for the application to gain access to this buffer for its own use in saving screen areas with raster copies. To do so, use:

```
wind_get(0, WF_SCREEN, &loaddr, &hiaddr, &lolen, &hilen);
```

Hiaddr and loaddr are the top and bottom 16-bits (respectively) of the 32-bit address of the buffer. Hilen and lolen are the two halves of its length.

Due to a preculiarity of the binding you have to reassemble these pieces before using them. (The actual value of WF_SCREEN is 17; this does not appear in some versions of the GEMDEFS.H file.)

If you use this buffer, you MUST prevent menus from dropping down by using either the BEG_UPDATE or BEG_MCTRL wind_update calls. Failure to do so will result in your data being destroyed. Remember to use the matching wind_update: END_UPDATE or END_MCTRL, when you are done. The other useful call enables you to replace the system's desktop definition with a resource of your choosing. The call:

```
wind_set(0, WF_NEWDESK, tree, 0, 0);
```

where tree is the 32-bit address of the object tree, will cause the AES to draw your definition instead of the usual gray or green background. Not only that, it will continue to redraw this tree with no intervention on your part.

Obviously, the new definition must be carefully built to fit the desktop area exactly or garbage will be left around the edges. For the truly sophisticated, a user-defined object could be used in this tree, with the result that your application's code would be entered from the AES whenever the desktop was redrawn. This would allow you to put VDI pictures or complex images onto the

desktop background.

A SIN OF OMISSION

In the last column, I neglected to mention that strings whose addresses are passed in the WF_NAME and WF_INFO wind_set calls must be allocated in a static data area. Since the AES remembers the addresses (not the characters), a disaster may result if the storage has been reused when the window manager next attempts to draw the window title area.

COMING SOON...

This concludes our tour of GEM's basic window management techniques. There have been some unavoidable glimpses of paths not yet taken (forward references), but we will return in time.

On our next excursion, we will take a look at techniques for handling simple dialog boxes, and start exploring the mysteries of resources and object trees.

Structure, Part 5
by Colin Masterson

What makes a good procedure?

We have spent some time discussing the basics of structured programs and shown that a good data type can be a help. We now go on to consider what makes a good procedure.

A good procedure

In 'C' we can pass as many parameters as we like - or none. We can return only one value of any type, or none. Functions can be 0 to 'n' lines long where 'n' tends to infinity. Function name lengths vary with the compiler but typically can be up to 31 characters long.

(i) bi(a,c,zz,q,t,ty,s)

(ii) locate_nearest_Eseries_value(res,series)

Both (i) and (ii) would be legal function specifications in 'C'. Which is the more readable ? Accepted, function names longer than more than 15 or so characters are a bit verbose - but look at the possibilities ! Hardly any comments needed.

In (ii) we have used a structure to gather closely related information about a data type. We pass our basic data and also a further parameter to control or determine the action of the function. This may be a simple TRUE/FALSE flag or a pointer to an array of values.

Passing a control parameter like this is a good idea. It allows you to code the function in a way which caters for future updates. It allows a single function to carry out different, yet related, tasks. It allows us to include optional conditions in the function which we can switch on or off according to how we call it.

e.g.

- annunciate out of range values.
- perform rounding upwards.

- display % error or remain silent.

A control parameter like this is often best placed first in the argument list. Why ? Well, in 'C', if we don't ask for other parameters then they don't have to be there. Thus, we could have a function which will display an alarm message and sound the bell:

```
annunciate(BELL,"Out of range.");
```

which we could call like this:

```
annunciate(CLR_MESSAGE);
```

Both of these work because the first parameter is a control. According to its value the function will or will not expect further parameters. Since none are required to clear the message we need only pass the control itself.

Furthermore, if we decided to add a complexity to our function which required further parameters to be passed, then we rewrite the function to only check for these further parameters if the control requires it. We do not need to change any of the existing programs which call this function.

(Note that LINT and similar systems may observe the fact that a function is called with differing numbers of arguments, it is, however, perfectly acceptable 'C'.)

Rather than passing a separate parameter to a function, another method is to set the data to some invalid, or out of range value.

e.g.

- -1 as a flag in a series of positive data.
- "" an empty string for a message function.
- NULL to a function expecting a pointer.

However, whilst these methods do have a place, I do not feel they are suited to selecting options within a function, nor are they suitable for choosing different paths.

NULL pointers can be a valuable way of detecting error conditions, and that brings us to the next topic.

The inside story

Having decided on the data type being passed to a function, a few general points can be made about handling it.

Validity: There will almost certainly be some checks you can perform on the received data. The only reason you might not perform these checks is that of performance - speed. This will normally only be an issue in low level functions which can be excused for not carrying out full validity checks. In medium to high level functions every opportunity should be taken to test for:

- presence of data.
- range limits. (Exit data too.)
- valid options specified.
- required globals present/initialised.
- prior process steps required, completed OK.

- error flags clear.

All checks wouldn't necessarily be included in all cases. The programmer must decide which to include. If you ask why you should include any when it's only you that's going to be using the function then the answer is that YOU make mistakes. An untrapped error may ripple through a program causing strange, sometimes invisible, errors in routines far removed. If a function is to be used by others and you don't put in proper error checking then it's like giving someone a cup of tea with a hole in the bottom of the cup !

If data appears not to have limits, or none have been specified, then you should feel free to apply some. A limit which is far removed from the normal working range yet still allows the function to operate correctly should be quite acceptable. All such boundary conditions must be clearly commented and documented.

Length: If a function has stretched to more than 70 or 80 lines then look at it closely for blocks of statements which form natural sub activities. No matter if this sub block is never called again - it is likely to improve the legibility of the program. Sometimes a function with multiple 'case' statements and lots of comments may stretch to 100 lines. This should be considered exceptional. A fine length for a function is one which fits on a single form of printout paper.

We'll continue next time with some talk about exit codes and function return types.

The Story Behind FLIT
by Colin Masterson

I describe FLIT as a 'multiple media disk copying program' which replaces the tedium of creating temporary directories and transferring files in and out of them when copying. This is the background of it's development with a look inside at some of the functions which may be of interest to DOS users.

The Background

I move around a lot. I work in offices, at home, offshore, here, there, everywhere.

The IBM standard has been great for me; WITHOUT IT, the type of work I do would NOT be possible; WITH IT, it's just a hassle.

The hassle part comes from the fact that I'm always moving files from one machine to another and copying disks from one format to another.

For a while I only had 5.25" disk to worry about; now I've got 3.5" ones too!

One thing that really drives me nuts, is having to sit at a machine waiting to open and close drive doors as I'm copying disks.

I had a dream; I dreamed of a program that would make my life a bit easier. I probed the public domain libraries, tested the tools, scanned the software and looked at lists; but couldn't find anything to do what I wanted.

That was the main reason for starting FLIT; I also had an interest in getting to know a bit more of the detail of the DOS/BIOS disk handling with a view to introducing some simple copy protection onto my own disks.

The Concept

The original concept was simple:-

Most machines have hard disks these days, all I needed was a program to:

- create a temporary directory on a hard disk,
- copy files into it,
- copy them off onto another disk and,
- remove the temporary directory.

This immediately revealed the need for a number of key 'engines' within the program.

These were:

- A disk format function.
- A fast disk copying function which would exactly clone every detail of a file.
- A directory copying function which would copy a complete directory and all subdirectories.
- A formatting function.
- A directory removal function to delete a complete directory leg.

The Plan

Like all good plans, I decided to take things in stages.

Of course, things turned out to be a bit more complicated than I had originally imagined. However, I think I'm there now and I've built some interesting functions along the way. I suppose a version of FLIT will be available in the club library.

To anyone who has ever thought about getting down into the nitty gritty of the DOS/BIOS disk level, my advice is DON'T! Not unless you really have to.

It's not so much that it's difficult, it's just that there is a limited amount of information around and much of it is unclear or contradictory. As a result, a great deal of experimenting is required. My hat goes off to the guys in companies like Central Point Software, MACE and so on. To get the level of knowledge these people have must take years.

With me, this sort of thing is almost a side issue. The timescales are in years but the knowledge level is a lot less!

At the application level, FLIT uses a library of window functions to handle the user interface. (This is available in the source library if anyone wants to play around with it.)

What the user sees is a menu of choices allowing him to set up the number and type of disks he is copying. FLIT looks after temporary storage areas and prompts for the disks as required.

Essential to me was the need to only ever have to insert a disk ONCE, regardless of its size and the number of copies being made.

FLIT tries to be sensible, as you would, about compressing several smaller sized disks onto larger ones. It will format disks if asked, or if it feels it's necessary. Sounds simple, and in the end the program is less than 80k bytes.

Where Do You Start ?
At The Bottom!

To begin with, I wasn't sure whether writing my own format functions would be practical. I started by unassembling the DOS FORMAT command and looking at the way it did things. From version 3 onwards, Microsoft use the IOCTL call, handling the disk in a general purpose device oriented way. IOCTL calls wrap the built in DOS and BIOS interrupts in a shell of control blocks, headers and so on. For my purposes this was too clever. Simple format interrupt calls would do me.

I used DEBUG to examine the details of the BIOS Parameter Block and boot records of disks formatted by programs like PCTOOLS, FORMAT and older versions of FORMAT.

By comparing this with the information available in books like the Norton guide and the technical reference manual, I was able to determine exactly what had to be done. I decided to proceed.

The Policy

I try to avoid the use of global variables as much as possible since these often lead to difficulties in understanding when others have to work with your code. It also restricts your ability to code and test modules in isolation. There are times when they are more or less essential. In general I have built modules relating to specific tasks. Within these modules I create a set of private static variables to hold relevant information. These variables may be changed by the use of 'posting' functions, or their current values be obtained through 'reporting' functions.

This aligns with the ideas of object oriented programming, we pass messages to objects (modules in this case) and ask them to do something for us.

In many ways this makes the code easier to read (with the right choice of variable names) and allows modules to be easily tested in isolation.

It is a reflection of the success of this method that most of the modules were tested with a very few lines of 'stub' code getting them to a high level of completion before linking them into the final application.

Engine 1 - remove a directory...count_files()

The directory removal function would form part of a useful stand alone utility (KD - KillDir). I decided to start here.

The resulting function count_files actually performs two roles (hence the title).

Given the name of a directory (including drive) it will either return the number of files in that directory (and all subdirectories), optionally listing or deleting them as it goes.

Now, the subdirectory structure leads one to think that a recursive function would be suitable here, and in fact count_files IS recursive.

However, we have to be careful. What we want is for DOS to name all the files for us and then, if a subdirectory is encountered, perform a recursive call.

Fact: DOS is not re-entrant, so when we use its directory search interrupt we must finish and be done with one directory before we can use the same interrupt call as part of any recursive 'C' function on a further subdirectory.

Since we have no knowledge of the order of files and directories on a disk, we have to assume that

we may be looking at a situation like this:

```
File1
File2
SubdirA-----+-- filea
File3  |___fileb
SubdirB        |___SubdirX----....
:
:
```

In other words, files and subdirectories appear in any order. This involves actually performing two scans of the directory at each level. One to locate the subdirectories and save their names, then another to perform the action on each subdir.

The 'action' calls may be recursive since we no longer have to 'hold over' a DOS interrupt into a recursive 'C' function call.

The pseudo code is:

```
If nesting level too deep
then exit
while a file or subdir exists
skip self and parent.
form wild card search pattern
if deleting read only files
        change attributes
if deleting
        delete all files in directory
if talking
        name the files
while any subdirs
        save subdir name
for each subdir
        form new subdir name
        recursive call
        if removing directory
        remove directory
remove current dir
return number of files counted
```

I put a check on the nesting depth because things can get out of hand if you're not careful. In real life, not many people go down to the 32 levels or so possible with DOS so I felt a restriction here was fair. It avoids the stack getting full since we need to grab quite a chunk from the stack each time.

Notice that the function to remove the files, do_deldir, uses the old style FCB call and not the new handle call.

The reason for this is that the old FCB call is actually much faster, since it allows the use of wild cards. There is no equivalent in the new calls as far as I can see.

There is a bit more fancy footwork in count_files() to handle read-only files, but not much.

With the careful use of the mode flag, the KILLDIR program simply adds a user interface, some protection and two calls to count_files(). The first call lists the subdirs and the number of files in each, the second call does the actual dirty work.

Be warned...!!

Notice the use of the word protection with reference to count_files().

I coded KILLDIR to place a number of restraints on its use since it's a potentially lethal function. It will quite merrily delete your entire disk without a whisper if you ask.

The protection I added in KILLDIR was:

- Deletion of the root prohibited.
- Deletion from another drive prohibited.
- Deletion of the current or a lower level directory prohibited.
- No clever stuff like ..\CURRENTDIR in the subdir name.

(KD is a fairly safe program to use unless you start adding the switches to force it to skip the listing.)

Now, although I have this protection in KD, while developing FLIT I had one or two nasty experiences because of the NUKE nature of count_files - BE WARNED !!!

I added one or two double checks on the calls but the biggest risk is that you'll ask it to select a directory which it never gets to but goes ahead and deletes everything anyway.

Engine 2 - fast file copy ...fast_copy()...

The fast copying function is fairly straightforward, it's necessary to use the DOS calls to get all the details about the file so we can exactly clone all its attributes.

Microsoft C makes this rather easy since they provide a set of DOS interface calls. For other compilers the use of the int??() style call or #asm directive may be required.

To make things as fast, but as flexible as possible, I have fastcopy try to grab the biggest buffer it can, gradually shrinking this until it gets too small to do anything. Fastcopy() returns an error code according to any problems it's had.

...and copydir() ?

The directory copying function of course calls the fastcopy function to do the work.

Copydir accepts the name of a source and target directory and clones one from the other...almost. It returns an error code according to it's success or otherwise.

DOS does not really allow us to create new directories with a date other than the current one. So, the date tagging on copied directories will not be the same as the original. (Does it matter?) The only way I can see to do this easily is to change

the date/time, call the the mkdir() function and then change the date/time back. This is not a nice thing to do! Also, it does not handle the fact that each level actually has it's own date &

time tag. I could have changed the methods to handle this problem but decided it was a fair compromise again. (Otherwise known as a 'slope shoulder' job.)

Some interesting points arise in this function.

What happens if the target disk get's full up ? We need to ask for a new one. Not just that, but this

might have happened three or four levels down in subdirectories, we need to create them on the new disk before we can proceed.

Also, in a friendly application, it's nice to see what's going on during lots of disk crunching and, if possible, allow the user to exit in a graceful way. (I just hate programs that don't forgive my mistakes)

The solution to the first problem requires creation of a function which will do what DOS doesn't let you do, create several nested subdirectories at once.

Create directory...create_dir()

create_dir() accepts the name of a path to build and parse out the names one by one, creating the subdirs as it goes. If it gets into difficulties it returns a fail code.

Although slightly restrictive on the passed string, this is a useful stand alone function.

And we might we expect that copydir() should be useful stand-alone - apart from the second problem we identified.

This opens up a question which arises (for me) time and again in application programs.

On the one hand we want to make these 'engine' functions as I call them, as general as possible - so we can use them on the next job. On the other, we need to tie back up to the application level somehow, to let it know how things are going.

The problem arises because we can see that copydir() may have to loop through several files, and possibly subdirs, before it returns.

The application may be text based, windows, file based – who knows? And why should the 'engine' have to know?

Informing the Application

We have several alternatives open to us.

1. We can make the variables that the copydir function uses external, and therefore accessible to other modules. We could then have copydir call an application level function to do whatever it wants with this externally available information.
2. We could call a function at appropriate stages and pass parameters with the current information. The application module could do what it liked with these parameters.

Both of the above methods mean that we must hard code the name of the function from the application module into our copydir function. This makes it less general and is something we may not want to do.

3. Alternatively, we could pass a pointer to a function into copydir(), then we don't need to hard code it's NAME, but we still need to know all about the parameters - so we are still hard coding here...little gain really (in this case).

What we really want is a way to control the execution of copydir, and get status from it, without changing the way it works.

In a multitasking system this is no problem but in a single task system like ours it's a bit more awkward.

Method 2 is chosen here, calling a function tell_copydir() as we copy each file. This passes the name of the file we are copying and returns true or false depending on whether copydir() should continue or not.

This is fairly satisfactory, if we're using copydir() in a situation where we don't need the interim reporting we can just code tell_copydir as a single return(TRUE) statement.

In FLIT, tell_copydir() is quite sophisticated, itself calling other functions to gather the information it needs to paint a full picture of the current status.

A Continuing Struggle

This same problem arises during formatting, the need to give some application feedback as to what's going on from a very low level machine dependent function.

There is a constant battle here to preserve the structured nature of our code whilst allowing user interruption and providing user feedback.

There is another way of handling this sort of problem which requires us to re-think the way we conceived the original task.

If you recall, we saw it as a loop, copying all the files until it was complete. We could also consider it as a function which copied the "next file in it's list" each time it was called. When one file is done, it returns a code reporting progress. Now, we still have to surround this function with a loop of some sort, the difference is that this loop is at the application level, and therefore has knowledge of the user interface. Before, our loop was at a low level with no knowledge of the user interface.

Of course, to do this may require a bit more complexity in our actual copydir function but this is a very useful technique.

In general, we could visualise an application program as being a loop, calling to each function in turn. If the function has nothing to do then it does nothing. It is up to each function to keep track of its own internal status. I have not used this method here although I have used it in more complex applications where the overhead is justified.

A Collection of Functions

There are one or two other functions worthy of a mention.

user_win - A general purpose, split window user input function. You pass details of the size and position of the window, the function to be called to paint the text at the top and bottom, and the function to be called to get user input. All the user input in FLIT is handled through this function.

pct_complete - A pretty moving bar display used to give the progress during a format operation. I must admit to having a liking for functions which are clearly contained within a single module which are called with a mode flag to determine the operation of the function.

Pct_complete() is a good example. It is a general purpose function which can be called with any current value and corresponding full scale reading, and will display the graphical pointer in a window. Since the function employs a window library to save patches of the screen and paint the image, it requires to be initialised before use and then removed afterwards. A flag passed to the function determines either, initiation, display or termination. The function uses good protective programming in that it checks that it has been initialised correctly before doing a display. This

function is also used to monitor the keyboard for an interrupt, returning the appropriate status to the caller.

horiz_list - Pass a window pointer and the list of choices the user can select from. The cursor keys or the first letter is used and the function returns the index of the selected choice.

Service Level Functions

Down at the service level there are a number of functions which will be useful for people talking to disks. The thing to remember about the format functions is that they must be called in the correct order. To format a track, everything, but everything has to be set up just right.

Volume Labels

Two functions, get_vol_label() and set_vol_label() allow you to do just that. They work on DOS versions down to 2 so can be used safely with most disks. They form a useful addition to your library.

Formatting

To actually format a disk there are several things you have to look after. These functions are contained in two modules FMTDOS.C and FMTBIOS.C. Since we are almost down at the hardware level, any thoughts of reuseability for these functions is abandoned.

We need to:

a) Set up the BIOS Parameter Block to be a table of details that the BIOS can pass to the disk controller IC.

b) Set up DOS so that it knows all the details about the disk, number of sides, sectors, tracks etc.

c) Prepare an image of address markers that will be written to the disk (used later for seeking to a given location).

d) Step through each track with a format & verify call to BIOS.

e) Build a File Allocation Table with a note of any bad tracks.

f) Write the boot record and the system tracks to the disk.

Who Knows What?

Notice that DOS doesn't know much about formatting of disks, it assumes that the address marks are available for it to manipulate files using the directory entries and the FAT. DOS has two functions, absolute read and absolute write to allow you to talk to disks in a sort of 'raw' mode. In this case I had no use for the read function but the write one might have been useful, the only thing is, I hate seeing the Abort Retry Ignore message thus necessitating writing my own absolute write function. This converts DOS sector number system to BIOS sector numbers and returns a code indicating success or failure.

Functions in Use

To use the format functions you first call disk_set() with the media type and the drive number. This sets up the parameters for DOS (set_disk_info), and BIOS (set_bpb_data), converting our high level name of the media to the number of sectors, system information and so on. disk_set() also performs any calls required to configure AT machines for the media size (INT17H call) and determines if the disk is formatted via the is_formatted() call. Finally, we determine if the disk is a system disk (DOS files in the first two directory locations) by performing the is_system() call. If it is a system disk, is_system() will have read the boot record which can now be posted to the system management module for safe keeping.

is_formatted() may be used after calling disk_set() to determine the format of any disk. It returns a numeric code according to whether the disk is formatted to the size we expect, whether it's blank and whether it's write protected.

Time to Develop?

Developing the actual format engine did not take as long as I though it might. It's broken down into sensible chunks and it's generally possible to test each one individually.

Getting the algorithms right to work out the cluster entry in the FAT, and to convert DOS sector numbering to BIOS numbering took a bit of playing with. After that, the judicious use of DEBUG to read back what's actually been written to the disk did the rest.

Even once the format engine was complete, and the user interface was clearly defined, I still spent quite some time making sure that all possible problems were handled in a sensible way. This is often the area that gets missed in functional specs since, very often, it is not clear what the impact of fault conditions will be until the system is actually running.

Hopefully, if the original methods and structuring are sensible, then coping with problems arising does not involve a complete rework of other modules. Having said that, there is no doubt that many functions go through an evolutionary stage where their return values or parameters are altered to suit changing situations.

Finally

The functions available allow you to format any kind of DOS disk, to read the type of disk (or if one is inserted) or to determine if the disk is an operating system disk.

This is the nitty gritty level I wanted to get to.

From here there a number of interesting ways to go but I rather suspect it's a bit too late to bother. Copy-protection has gone, optical disks are not far off, the day of the media ID byte has almost gone forever - perhaps the next standard will be consistent in the way it uses things.

The Source Library
by Martin Houston

What is it all about?

INTRODUCTION

The CUG library is intended to be a collection of public domain C source code brought together so that anyone who wishes to may use the programs and further develop them both as an aid to their own learning and enjoyment and to improve the stock of C software in the Public Domain.

Very few real programs are ever written from scratch. Most things are developed using and adapting parts from previous programs. The library will serve as a 'toolkit' of parts to aid in the process of developing new ideas.

May PD Software & 'Shareware' libraries carry some C source code amongst their wide selections but the CUG library is specially dedicated to making source code available for programmers to develop with rather than being a source of 'ready to run' free programs. This feature makes the library useful to people with a wide range of machines and interests.

If you have a machine with a C compiler that is able to read one of the disk formats the library is offered on then we have something to offer you. Many other library services cater only for the strict IBM PC clone market. The CUG library does not care if you have a PC, an Atari, an Apricot, an Archemedies or even a Unix system.

The two disk formats that the library directly supports are the PC 360k standard for 5 1/4" disks and the 720k double sided 3 1/2" format used by IBM PS/2, Apricot, Atari and most of IBM compatible portables. Single sided variants of the above formats are also available if requested specifically.

To read a library disk you will need to be able to understand Microsoft MS-DOS disk format WITH sub-directories i.e. DOS 2.0 or above. The library disks will use sub directories to partition files into groups that belong together. If a program has more than one source file it will generally have a directory to itself unless it is obvious which files belong together. No form of archiving or packing will be used on the library disks. This removes the need to have a version of ARC (or similar) that runs on your machine. One exception to this is that volume 14, "Larn" is Squeezed so that the files fit onto a 360k volume. Source of the unsqueeze program is provided so that it should not take much effort to get at them. If you want to download software off one of the online systems CUG sponsors then you will need an ARC program as the library files will be available online in .ARC format to save space & download time. Please do not expect to have the whole library available on a BBS. It is up to the SYSOP to decide how much disk space he is willing to spare for CUG library use.

If you cannot cope with an ARCed file then a kind message to the SYSOP of the board may result in the files being left un ARCed for you to download individually.

Except for a few documented exceptions everything in the library must be in source code form. The library will only carry 'shareware' function .LIB files on disks that are indicated to be for one type of machine only. If a library disk contains material that can only be used by one type of machine this will be indicated in the library list. Most library disks will be usable by any machine (with adaptation in some cases).

For some programs an MS-DOS .COM or .EXE version of the program is included. You run this

program at your own risk!!!! - In most cases it is the executable that was given to the library at the same time as the source but I cannot guarantee that it works/is the same program/is not malicious (a Trojan). Some but not all of the .EXE files have been generated from the sources by myself.

The library does not at present cover the Apple Mac or any of the CP/M formats (including Amstrad P.C.W.). If you have one of these machines then contact me and I can arrange transfer of the volume of your choice by modem. This will be for the same charge as the disk would be but you will also have to pay the phone bill.

Another alternative would be for you to find a friend with a machine that can read one of the disk formats and do a serial transfer over a direct RS232 line.

HOW IT WORKS

The way the library works is that any C code donated to the library will be catalogued and a CUG Library header comment prepended to the file. This comment (reproduced below) is intended to allow the change history of the file while in CUG hands to be recorded. It is hoped that any CUG member that improves a library program will re-submit it to the library so that all may benefit from the work they have done. In this way all may learn together and produce good software for the benefit of all. Here is a sample comment (anything between <and> is

variable information):

```
/*************************************************************
 * C Users Group (U.K) C Source Code Library File <disk num>        *
 * Inquiries to: M. Houston, 36 Whetstone Clo. Farquhar Rd.         *
 * Edgbaston, Birmingham B15 2QN ENGLAND                            *
 *************************************************************
 * File name: <name of this file>                                   *
 * Program name: <name of program to which file belongs - which     *
 *               could be a library or an executable>               *
 * Source of file: <where the original came from before the library>*
 * Purpose: <what it does>                                          *
 * Changes: <who what when & why major changes have been made>      *
 *************************************************************/
```

<disk num> is the library disk number that the file comes from - 0 means that the information has not been filled in.

The library will try to ensure that each file that goes out will have at least a blank header but it is largely up to the members to fill the headers in and keep them up to date. I feel that this is a bit of discipline will be of great benefit to the CUG members as it will enable a database of what is in the software library to be kept so that answers can be supplied to such questions as "Can you find me a function that does pattern matching?" or "Are there any assemblers in the library as C source?". It will take a while before the library enquiries database is fully operational but I feel it is a goal well worth attaining. The header will usualy only be put in the 'main' module of any multi module program that obviously belongs together (such as the xlisp sources). If members want to go through and header each and every file then they are welcome to but doing so intelligently requires considerable knowledge of the structure of the program. The poor librarian cannot be expected to be an expert on every program in the library!

LIBRARY LISTS

Each issue of C Vu will contain a list of what is in the library. In addition to this a separate totally

up to date list is available to anyone who sends a large S.A.E together with 20 pence in stamps to cover expenses. The library list is also available for download from the Chronosoft BBS or Dr Solomons FIDO (details in C Vu). If you are looking for something in particular then contact me and I shall try to find it for you.

LIBRARY ORGANISATION

Each library disk will be dedicated to a specific subject area as the range of material in the library permits. For instance material that is strictly of interest to people with IBM PC compatible hardware will be isolated from generally portable material. This division is made on the barest aquaintances with the program - I cannot guarantee that everything in the generic section is suitable for all machines – the specific machine divisions are for programs that are OBVIOUSLY only of interest to owners of a specific machine.

When a library volume is strongly suited to one particular type of machine in this way the list will say so to save people ordering disks that are of no use to them. Needles to say the C Users Group can offer no warranty as to the quality and fitness for any purpose of anything in the library. Public Domain programs are almost bound to have bugs in them; it is hoped that the work done by group members on the software that comes into the library will improve this situation.

UPDATING THE LIBRARY

The initial offerings in the library have mostly been culled from the offerings of other 'shareware' libraries & BBS downloads. Some material has been donated by CUG members. At the moment organisation of the library is in its early stages and the quality of the software patchy. I have endeavoured to cut out on duplication and the banal leaving a reasonable base of material on which to build a strong UK Public Domain resource.

If you have taken one of the programs from the library and have made significant improvements to it then you may re-submit the improved version for inclusion to the library. Any improved versions of the library programs will be made available to the membership as separate volumes will full acknowledgement to the member who has done the improvement work. The original version will remain available as the original volume number. In this way the development of the programs under out control can be traced and the same program can be allowed to branch off into many separate development paths. Anybody submitting an improved version of a library program will be entitled to an equivalent amount of new library material at no charge.

If you indicate that you would like to be put in touch with other members that are working with the same program then this can be arranged through the pages of C Vu (a sort of programmers dating agency!!). Some of the programs already in the library such as the xlisp lisp interpreter and bawk text processing language are complex enough to need some joint effort to get to grips with them.

ORDERING

Library material is available to CUG MEMBERS ONLY and the standard rates are shown in the “Source Library” listing. Specially selected material incurs a £4.00 surcharge on the standard rates.

PLEASE NOTE that if you want a single sided 3.5" disk then please say so when you order. The 3.5" drive on my XT machine where the library is kept will not format single sided disks so any single sided orders have to be processed through my Apricot F2. I know that Ataris come with a single sided disk as the cheapest option but I should think that anyone thinking about C

compilations would have at least a double sided drive. Am I wrong? If so please tell me.

The surcharge of 4 pounds is made for a library order other than one of the standard prepared disks. When the data base system is in operation it will be possible to look out a selection of files matching certain criteria. This takes time however so the surcharge is made. The surcharge will apply to every multiple of 360k of data selected. I hope that my efforts in sorting out the library volumes will mean that no one will ever need the special service!

If you are donating material for the library (or are returning an improved version of a library program) then you may have any standard volume from the library copied on to the disk for no charge. (The Librarian reserves the right to judge what is a sensible contribution to qualify for a free volume - sending a disk with 'hello world' on it will not get you off paying the two quid!).

CONCLUSION

I hope that this has given you an insight into how the library is intended to work. Hopefully the library will become the focal point of the activities of the group as there is much in it as topics for discussion and development. I hope you find the charges are reasonable; the library takes a lot of time & machine resources to run effectively so it cannot be a free service. Two pounds for over 300k of C source code & related material is very good value for money and goes some way towards contributing to the running costs of providing the library.

One final word: the library is only going to be any good if it is USED. If it is used well with people contributing as well as taking then it could grow into a valuable resource for the C Users Group & all C programmers.

The Dreaded Pointer! Part 1
by Phil Stubbington

The cause of nearly all bugs - Phil Stubbington investigates

Introduction

When I was first trying to gt to grips with C, pointers were the feature of C that caused most problems. What were they, why were they there, are they the same as pointers in Pascal? Why did my programs crash spectacularly whenever I started trying to use them? Then, one day, a sudden flash of inspiration! (I try to make the most of those – they don't happen that often). The purpose of this article is, hopefully, to reduce the “bashing head against brick wall” phase. Let me know if it succeeds!

The Starting Point

Right, first things first. I'm assuming that you already know the basics of C - you can quite happily cobble together a program off your own back. In particular, you should already know about the various types of variable: char, int, long, float and double. You should also know how and where they are declared (auto and static classes). Equipment wise, I'll assume you have access to a C compiler, linker, etc. and appropriate documentation. The example programs to accompany this series have all been written using the Prospero C system, but should work (perhaps with minor mods) with any C compiler which has leanings towards the ANSI draft standard.

What Is A Pointer?

This is no doubt going to sound incredibly obvious, but a pointer is a variable able to hold the address of a C object (either a variable, or a function). There are two types of pointer - explicit and implicit. For the time being III ignore implicit (which are compiler generated, to handle arrays) and concentrate on the explicit ones (i.e. ones which appear as a result of declarations made by you in the program).

Once a C program has been written, compiled into machine code, and linked with any library functions it needs, we can execute it. Depending upon the operating system being used, it will either be executed from a fixed location in memory (unusual) or from wherever the OS can find sufficient contiguous free space to load the program into.

Naturally enough, as well as the machine code that is actually executed, the size of the program file includes space for any variables that you have declared. So each variable will take up specific locations or addresses in memory. How many locations they take up is a factor of their type. In some implementations, an 'int would take up two bytes, a 'long four bytes, and so on. The details of this will (or should) be, in your documentation.

If you type in, compile, and run the following program it will display the addresses of the variables count and increment together with the contents of those addresses:

```
/*                                              -*/
/* P0INTEX1.C                                    */
/*                                              -*/
/* Objective:                                    */
/* o use of '&  (address of) operator to find    */
/*    location in RAM of variable at run-time    */
/*                                              -*/
/* Compiler: Prospero-C                          */
/* Machine: Atari ST                             */
/* Reference: Article  The Dreaded Pointer ,     */
/* Part 1, By Phil Stubbington                   */
/* C Vu, Volume 1, Issue 5. July 1989            */
/*                                              -*/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

void main()
{
  int Count,Increment;

  assert(sizeof(long) == sizeof(int *));
  Count = 32;
  Increment = 97;

  /* +                                  -+
     | Using '& (Address Of) Operator    |
     | To Find Location In RAM Of        |
     | Variable At Run-Time              |
     +                                  -+  */

  printf("'Count = %d\n\tAddress of 'Count  - %lX\n" ,Count,&Count);

  /* +                                  -+
     | Same again!                       |
```

```
      +                                  —+  */

  printf ("'Increment = %d\n\tAddress of 'Increment = % lX\n",
          Increment, &Increment);
  puts ("Press Return/Enter key to exit");
  getchar() ;
}
```

So, having run the program you might get the results:

```
Count = 32
  Address of 'Count = 25712
Increment = 97
  Address of 'Increment = 35710
```

Graphically, this would look like Figure 1.

The Address Of... Operator

In POINTEX1 we used the ampersand (&) which will tell us the address of a variable. This in itself is not terribly useful, but we have to start somewhere! One thing to note is that 'auto class variables are local – i.e. they effectively vanish when the function terminates, so having a 'static (global) pointer to an 'auto (local) variable is a sure recipe for disaster!

Why Use Pointers?

Now we know what a pointer is, we come across the slightly more difficult question - why use them? Three reasons really:

- Speed. If you've ever dabbled in assembly language, then things like address register and indirect addressing are two terms you II no doubt have come across. Do remember that C is really only one level above assembly language in the machine accessibility stakes. Many things you do in assembler have a direct equivalent in C - and pointers are the nearest thing to address registers.
- Accessibility. We've mentioned that pointers contain the address of a memory location. That memory location could well be the control register for a hard disk controller. So, if the mood, or need, takes you, C can be used to control a multitude of devices without having to drop into assembler.
- Tina. Yup, There Is No Alternative! As we II discover later on, there are some things you can t do without messing around with pointers. One of these is using dynamic memory. If the exact memory requirements of your program aren't known at compile time, then you can t really avoid using pointers. An obvious example is the text editor or word processor. You don t know at compile time what size the file is the user wants to load, and you don t know how many lines they II add and delete after it is loaded.

Pointer Declarations

In our second example, we are finally going to introduce a pointer. Like any other variable, we have to declare it before we can use it. A pointer, again like any other variable, has a name and a type. In this example we declare a pointer called ptrToInt. We know it is a pointer because the name is

prefixed by an asterisk. We also know that it is a pointer to an 'int, because that is the type given to it. So, the declaration is best read from right to left. We are declaring a variable called ptrToInt, which is a pointer to an 'int. So, the object that ptrToInt points to is 'int sized (probably two bytes), but ptrToInt is pointer sized (probably four bytes - i.e. sufficient to hold a machine address). The full declaration is:

```
int *ptrToInt;
```

Note that, as usual, because it is an 'auto class variable in the example, it is guaranteed to contain rubbish unless we initialise it.

Initialising A Pointer

In the example, we have initialised ptrToInt so that it contains the address of the variable called count. As before, we find the address of a variable by using the ampersand:

```
ptrToInt = &Count;
```

Reading A Pointer Variable

We have now initialised the variable ptrToInt. Again, as with any other type of variable, we can assign a value to it (as we've just done) or read the address back out again:

```
printf("ptrToInt = %1X ......... \n", ptrToInt, ........);
```

Reading What The Pointer Points To!

Finally, we can find out the contents of the address that the pointer is looking at:

```
printf ('',.....Contents of
ptrToint = %d\n  ,.......,*ptrToInt);
```

Note the subtle difference. Just referring to the pointer name alone yields the address. Prefixing the pointer name (outside of the initial declaration) with an asterisk yields the contents of that address.

Before you compile and run the following program, write down on a piece of paper what results you expect to see. Obviously, unless you re a mind reader, you won t know the actual run time addresses (and that isn t the point of the exercise anyway!). So just make a wild guess.

```
/*                                                    -*/
/* P0INTEX2.C                                          */
/*                                                    -*/
/* Objectives:                                         */
/* o declaration   of a pointer                        */
/* o use of '&  (address of) operator to               */
/*   initialise a pointer                              */
/* o use of pointer variable name to either            */
/*    assign value to, or read contents of,            */
/*   pointer variable                                  */
/* o use of '*  (contents of) operator to read         */
/*    contents of object that pointer                  */
/*   points (!) to                                     */
/*                                                    -*/
/* Compiler: Prospero-C                                */
/* Machine: Atari ST                                   */
/* Reference: Article ' "The Dreaded Pointer           */
/* Part 1, By Phil Stubbington                         */
```

```
/* C Vu, Volume 1, Issue 5. July 1989                  */
/*                                                    -*/
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
void main()
{
  int Count,Increment;

  /* +                                    -+
     | Declaring A Pointer To An Integer  |
     +                                    -+ */
  int *PtrToInt;

  /* NOTE: as PtrToInt is an 'auto  class variable */
  /* (as are Count and * Increment) it contains    */
  /* gibberish until we initialise it              */

  assert(sizeof(long) == sizeof(int *));

  /* +                                      -+
     | Using Pointer Variable Name To        |
     | Assign Value To Pointer Variable      |
     +                                      -+
     | Using '&  (Address Of) Operator To    |
     | Initialise A Pointer                  |
     +                                      -+ */
  PtrToInt = &Count;

  /* now PtrToInt contains the address of Count, */
  /* i.e. it points to it                        */

  Count = 32;
  Increment = 97;

  printf(''Count = %d\n\tAddress of 'Count = %lX\n ,Count,&Count);

  /* +                                      -+
     | Using Pointer Variable Name To        |
     | Read Contents Of Pointer Variable     |
     +                                      -+
     | Using '*   (Contents Of) Operator     |
     | To Read £ont.enta ~i Object That      |
     | Pointer Points (!) To                 |
     +                                      -+ */

  printf(''PtrToInt = %1X\n\tContents of 'PtrToInt  = %d\n,
         PtrToInt,*PtrToInt);
  puts (''Press Return/Enter key to exit'');
  getchar();
}
```

If you now compile, link and execute the program, what results do you get?

What you should have got is something like this:

```
Count = 32
  Address of 'Count =35718
PtrToInt = 35718
```

```
Contents of 'PtrToInt  = 32
```

In other words count is 32 (no great surprises there, I hope). The address of the variable Count is x. The value in the pointer variable PtrToInt is also x. Finally, because PtrToInt contains the address of count, the contents of that address should be the same as the variable Count - i.e. 32.

Is That It?

Well, no it isn't! What we have covered in this short article are some of the fundamentals of pointers. We've seen that we can use the address of operator (&) to find the address of a variable. We've declared a pointer by telling the compiler what type of variable the pointer points to, and giving the pointer a name (prefixed by an asterisk, to indicate that it is a pointer). We ve seen that using just the pointer name can be used to either assign an address to the pointer or read the pointer address itself. Finally, we've seen that prefixing the pointer name with an asterisk (outside of the initial declaration) can be used to read the contents of whatever the pointer points to.

If you d like to see more, or fewer, articles, like this then drop us a line at the usual address, or I can be reached on CIX as 'philip'. Similarly, if you dropped off half way through, and would rather things went a bit quicker, or conversely got lost after the first paragraph - LET US KNOW! Do bear in mind the old 'you can t please all of the people... proverb -I'II go with the majority decision!

One Person's View
by Francis Glassborow

Francis Glassborow makes his debut in C Vu – and gives out a few brick bats amongst the bouquets!

Sometime in the back end of last year I began to wonder if I had thrown my money away when I joined CUG. Then a copy of C-Vu dropped through my letter box. As it had been such a long time since the previous issue I wondered if I had missed an issue. I wrote to Martin Houston expressing some concern and at the same time offering a disc of C source code for the library. The up-shot of that letter was a phone call and eventually my attendance at the first AGM of CUG(UK).

By the time you get to read this many months will have passed since your last issue of C-Vu. However, you would be mistaken if you thought that CUG was slowly burying itself in inertia. Though only a dozen members attended the AGM it was undoubtedly the most important event in the history of CUG. A committee was formed which, by now will have met twice and is already dramatically developing ideas for the future of CUG. As from September the ordinary member should begin to notice a steady increase in activity. On the other hand you the members should expect to increase your own activities.

About now you may be wondering why I am writing this. That's difficult to explain as I am only a simple committee member and not one of the officers. But I think that promises of changes and goodies to come might be better reported by an erst while ordinary member who has had his moments of doubt. In the rest of this article I will describe a few of the things that I know about and hope you will wish to support (actively for preference).

The PC Show (ex PCW Show)

You know the one that used to be called `The PCW Show'. We are going to be there this year and if

you are coming please help in the following ways:

1) Find our stand in the ACC section when you first arrive and introduce yourself.
2) Collect, and wear a badge declaring your CUG allegiance so that others will see that we are alive and healthy.
3) Volunteer to help on the stand even if it is only for half an hour.
4) Come to the daily meets that we will be arranging.

It is your committee's opinion that we should be doing everything possible to increase the amount of dialog and social intercourse between C Users. The PC Show activity is only a small part of this.

Local User Groups

Up until now the only form of membership of CUG has been for individuals. As from September that completely changes with the introduction of Corporate or Group membership. The idea behind this is to get C Users meeting, talking and providing mutual support and encouragement. There will be a single corporate membership fee that is independent of the number of people making up a group.

During the summer I will be putting together a package of material for use by members who want to start their own local group (and we hope that you all will). This package will be based on the experience of the Oxford and District C User Group which grew out of the very successful C Programming Workshop that is run by the North and West Oxford Community Education Committee. The package will include:

1) Advertising posters and other promotional material to help you get started.
2) A draft constitution (just something simple to keep you legal)
3) Suggestions about who to contact for help in all aspects of developing a local group.
4) Ideas on running a programming workshop
5) Running your on branch of the National CUG source code library.

If you have any other ideas about things that would help in getting a local group started please contact me so that they can be included. The pack will be ready for the PC Show but earlier notification of interest would be welcome and encouraging.

C Programming Workshop

From September 1988 till March 1989 I ran a programming workshop on monday evenings at a local school. About a dozen people turned up regularly with a few more casual attendees. Gratifyingly most have already expressed the intention of joining again this Autumn.

The range of participants was very wide including the young teenage enthusiasts at one end to a retired enthusiast with over thirty years of programming experience. For one member it was his first experience of using a computer for anything.

As it was a workshop, not a course, the range of experience and background only served as an advantage. Technically I was in charge but very soon we had established a pleasant working atmosphere in which people were freely helping each other. The willingness to help the novice is extremely important and needs to be nurtured if CUG is to avoid the fate of several other user

groups that have become small cliques of `experts'.

At the end of this article (? collection of odds and ends) I have included a piece written by one of the less experienced members of the workshop. I think that the infectious enthusiasm of the young coupled with the expertise of the elder statesman has done much to ensure that Donald has enjoyed his monday evenings and is looking forward to continuing next year. I hope this encourages some of you to plan the same sort of thing for your own area.

A Bibliography

C Programming, bibliography and information pack, Paula D Fountain, 1988, IEE P.O. Box 26, Hitchin, Hertfordshire, SG5 1SA. Price £29 pounds

Those of you who read Byte may have seen a notice of a package produced by the Institute of Electrical Engineers. They have kindly sent me a complimentary copy (well I have a couple of contacts). I think I should warn you that it is only useful to those who need a rapidly outdating listing of articles, books and compilers. Unless you have experience of this kind of material you will be very disappointed. The entire package is a single A4 booklet of less than 50 pages of material produced by a word-processor that fully justifies text by inserting whole spaces. Honesty forces me to query the utility of such packages when a continually updated package devoid of the glossy cover would prove to be far more useful.

APDCCWFSCFTAST!

A Public Domain C Compiler, with full source code, for the Atari ST!

Oh you lucky Atari ST owners, a complete public domain C Compiler called Sozobon has just come the way of CUG. It includes the full documented source for the compiler, assembler, optimiser and linker together with a library manager, libraries (everything includes the relevant source code) and the most open Public Domain release note that you could want (well you musn't sell it but you are positively encouraged to pass it on and develop it).

Of course it has its weaknesses. There is some doubt about its handling of functions passed as pointers in parameter lists, and its floating point routines are rather minimal (no trig and double is only a synonym for float). But every local group should have a copy and we will be looking out for updates from the original authors.

An Amiga expert shouldn't take long to modify it to run on that machine. Ron has already tested it as a cross-compiler on a PC so as soon as an Amiga owner comes up with the relevant source mods for that machine a version can be compilled and ported for that part of the C community.

Anyone else who is interested in getting to grips with compiler writing will find Sozobon an excellent place to start. If we can find the right enthusiast with enough time we might even manage to produce an ANSI standard PD compiler and then prehaps follow with one for C++.

Don't forget that it can compile itself so you can steadily work at improving what is already better than a number of the sub fifty pound offerings. It will cost you very little.

Note that the libraries are in source code so that these will be of more general use and interest and not much will need to be done to get the library manager to function with your favourite compiler. The professional C programmer may wonder why I am raving about a product that is clearly inferior to their professional development systems. The answer is simple, this is an entirely open package at a cost that even the poorest of enthusiasts can afford.

Winding down

A word to the experts, everyone knows something you don't and once you knew nothing so be patient with the ignorant.

A word to the ignorant, the experts once knew less than you do, listen to them with respect but do not let then overawe you. One day you will know something they don't.

Well thats it from me. Why don't you have a go. Tell us something about your experience of C. Tell us what you use it for. Tell us what you would like to know about. Send in your favourite function. Whatever you do, don't just sit there and hope someone else will provide something for you to read on the next wet afternoon. If you do you will simply have to put up with more from me and the rest of the ODCUG lot.

All the best for now, hoping to see lots and lots of you at the PC Show this Autumn.

The Source Library

The Source Library is intended to be a collection of C source code in the public domain, brought together so that any member of the C Users' Group (U.K.) can user and develop it as an aid to their own learning and enjoyment, and to improve the stock of C source.

1 **Software Tools #1** - bracket and comment checkers, structure analysers, three variations upon the XREF theme, several hex loaders and dumpers, time command execution...

2 **Games #1** - principally the AdvSys adventure writing system with all sources, documentation, and a sample adventure. Also contains the Towers of Hanoi and Conways' Life

3 **Editors** - one of the latest version of the MicroEmacs editor; with sources and extensive documentation. Now contains a macro language and several other enhancements

4 **Languages #1** - language compilers and interpreters in C. Thus volume contains sources for XLISP; the object-orientated version of the lisp programming language.

5 **Math** - a comprehensive collection of the usual functions. Also contains a set of MASM macros for the 8087 co-processor, programs for numerical integration and matrix manipulation

6 **Unix Utilities #1** - pattern matching language, "make" project management tool, stream editor (complete with example sript - convert Pascal to C!) and several minor utilities

7 **Unix Utilities #2** - text processing and printing; a couple of text formatters, entabbers and detabbers, menu driven setup programs for IBM and Epson printers....

8 **Shells** - two Unix style (single-tasking only though) shells for MS-DOS - source code included so conversion to the ST & beyond is a possibility - why not give it a try!

9 **C Language Tutorial #1** - this disk contains the text of the tutorial. Most of the example programs can be found on volume 10

10 **C Language Tutorial #2** - the example programs to accompany the text found on volume 9

11 **Comms #1** - several comms protocols (Kermit, SEAlink) and library managers (Arc, Lump, Squeeze) and some related utilities

12 **PC Utilities #1** - some of which are provided as executables only. Contains an extensive shareware window manager, cursor control progs from Bill Sparrow, EGA graphics routines,.....

13 **Languages #2** - two subset C compilers (cpcn and ratc) with sources, an interpreter (sci) as MS-DOS executable only, and a number of Unix style utilities.....

14 **Games #1** - the Dungeons and Dragons style game "Larn". Should be possible to get it running on most systems. Requires lots of memory to run!

Please remember that double volumes must be ordered together, and will cost twice as much. At present this covers volumes 23a & 23b, 24a & 24b and 25a & 25b.

Your own contributions are always welcome!

15 **Unix Utilities #3** - source for LEX (lexical analyser) for MS-DOS, but should be possible to convert to other OS's

16 **Unix Utilities #4** – lots of material related to “lex” & “yacc” (yet another compiler-compiler)

17 **Unix Utilities #5** – another parser generator like “lex” called Bison. Source is included, but documentation is minimal, so some detective work is called for. Any offers?

18 **Software Tools #2** - some more programmers tools (see volume one) and a simple Unix Curses type screen library

19 Minix #1 - conversion of the Kermit comms program. Conversion and documentation (of use to any Minix devotee) by Adrian Godwin

20 **Unix Utilities #6** - mainly a healthy bunch of benchmarks, but also a number of general purpose utilities

21 **Games #2** - the one that started it all off - the Colossal Cave adventure with full source code

NEW! 22 **Minx #2** - Adrian Godwin has been busy again! How to build C-Kermit, fixes & library changes, a C-Kermit executable (use 'dosread', rename to 'kermit' and chmod to 755), and a Minix 'more' & 'ls'

NEW! 23a **Languages #3** - Small C/Plus for the Z80, by Ron Yorston. *Volumes 23a and 23b must be ordered together.* See the article in this issue.

NEW! 23b **Languages #3** - Small C/Plus for the Z80, by Ron Yorston. *Volumes 23a and 23b must be ordered together.* See the article in this issue.

NEW! 24a **Comms #2** - Sources for Kermit. This is a FULL version suitable for a wide range of machines. In particular it is suitable for MINIX. *Volumes 24a and 24b must be ordered together.*

NEW! 24b **Comms #2** - Sources for Kermit. This is a FULL version suitable for a wide range of machines. In particular it is suitable for MINIX. *Volumes 24a and 24b must be ordered together.*

NEW! 25a **Comms #3** - Docs for Kermit. Essential to make full use of Kermit. *Volumes 25a and 25b must be ordered together.*

NEW! 25b **Comms #3** - Docs for Kermit. Essential to make full use of Kermit. *Volumes 25a and 25b must be ordered together.*

The C Users' Group (U.K.) makes no representations or warranties with respect to the contents of "The Source Library" listing, and specifically desclaims any implied warranties of merchantability or fitness for any particular purpose of the library volumes mentioned herein.

NEW! 26 **Games #3** - Maurice Watson's QL version of Volume 21 - 'Colossal Cave'. See the article in this issue.

NEW! ?? Why not contribute some of your own work, or conversions of existing library volumes to other systems? Non-machine specific contributions especially welcome.

Order Form

To order any of the volumes in The Source Library simply circle the appropriate volume numbers in the space below (you can photocopy this form) and send it with the correct payment to:

The Source Library, C Users' Group (U.K.), 36 Whetstone Close, Farquhar Road, Edgbaston, Birmingham, B15 2QN

Please send me the volumes I have circled below

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23a	23b	24a	24b	25a	25b	26	

____ volumes at £2.00 (sending your own 5 1/4 or 3 1/2 disks) : ____:00

____ volumes at £3.00 (on our 5 1/4 disks) : ____:00

____ volumes at £3.50 (on our 3 1/2 disks) : ____:__

TOTAL : ____:__

I enclose a cheque/postal order for £____:__ made payable to C Users' Group (U.K.)

Membership Number: _____ *Signed:* _____

Why Programming? Why "C"?
by Donald Wilcock

I had bought my computer, which was to be my belated introduction to the mysteries of computing, in April. By the time that it came to September, obviously I still did not know very much about it. It is difficult to say, now, exactly what I expected to gain from some knowledge of programming. I think that I have come to believe that the whole concept of computing is something like a drug - the more one has of it, the more one realizes that there is so much more to know, to find out about; one is drawn onwards towards wider and ever more time-consuming horizons. Programming was one of these directions: what was it that could make the computer respond with unbelievable alacrity or with considerable reluctance, according to its whim, to some arbitrary command?

I remember standing in front of the bookshelves at Harrods where the computer books are and looking at the introduction or back cover of every book that announced by its title that it touched upon the subject of programming. I soon came to the conclusion that the "C" language was the thing of the future. Whilst searching for some formal instruction in "C", I came across a "C" workshop, for beginners and more experienced programmers, which was shortly to start in the school opposite to where I live in Oxford.

After two terms and six or seven months later, I can take stock of the methods employed and try to evaluate where I stand now and where I want to go. I can remember that on the very first evening my impression that everyone engaged in the computer industry was either a Yuppie or a rogue (or both) was dispelled by the down to earth attitude and genuine helpfulness of everyone present. There were a good number of young people - 15 year olds seem to have a head and a half start upon people like me - and it transpired that some of the older people had done programming in other languages before. One man was a real expert. Our tutor gave a lecture on pointers for about half an hour. In this half hour he also introduced us to more of the esoteric words: arrays, variables, strings, #include, printf etc. He handed out copies of what seemed to be a fairly complicated programme and said that if we did not have anything already, we should work through this programme. There was a pile of books on one of the tables for consultation. So we all went over to the computers and started. It seemed that day that I had a question about practically every single word or symbol. Francis, our tutor, gradually worked round the class giving helpful advice and suggestions so that we all, in our turn, could see our way to one or more steps forward. This evening set the pattern for the weeks to come.

By the second class I had bought a book and I worked on one of the examples from it. By now, it was obvious that I needed a compiler of my own so that I could work at home. Francis was very helpful in answering my questions about this and when I had bought one, how to actually get it to work on my machine. As the weeks went by I had an idea of what I wanted to create as a programme of my own. I worked on this at home until I was stuck and on the class night I would set up the programme on the computer as far as it went and when Francis came round, he would show me where I had typed it wrongly or make suggestions about changes which would advance it to its next stage.

By the end of the second term, I had a working programme, about three screensfull of "C". When run, this asks 3 questions, which, if answered correctly, each allows the operator to proceed to the next. This has proved to be a never ending source of amusement to every visitor to our house, most

of whom use computers at work and, incidentally, most of whom have never heard of "C". One other very noticable effect of the winter's work is that I now approach the task of operating my computer with much more confidance. Some operations, which earlier would have been a source of great difficulty can be performed precisely and quickly and with a far greater expectation of success.

When we recommence the class in the autumn, I would like to work on a programme which would really be of use to me in my everyday work - a spot of DIY programming, if you like. I think that I have some idea about what is possible and my work on the previous programmes has given me some clues on the ways I might go about it.

Scientific Magic Circle?

That's what the newly formed Scientific PC User Group aims to be - a independent, unbiased resource dispelling some of the myths and fears that surround scientists and engineers for locating help, information and advice for scientists and engineers.

Most of the popular software and hardware products are supported - including all versions of Asyst/Asystant, Labtech Notebook, Labwindows, the inevitable Lotus products (123, Symphony, Hal, Measure and Manuscript), MathCad, Labview, etc. Language wise, the group supports 'C' in all its' incarnations, Forth, QuickBasic and Assembler. Finally, on the hardware side, products from Analog Devices, BioData, Hewlett Packard, IBM, National Instruments, Siemens and TecMar all get a mention.

The main activities of the group are very much in line with ours - a newsletter (future articles are to cover everything from an introduction to C, through transient capture and time series analysis, to useful Lotus 123 macros and getting the most out of co-processors), local meetings, public domain software, applications, and details of new and enchanced products.

Individual membership of the group is £14.00 p.a., corporate membership is £45.00 (for up to five names, and an extra £10.00 per additional name). Added to each membership is a once-only registration fee of £4.50.

If you would like further details of the group, just fill in the questionnaire included with this issue, or write to them at P.O. Box 17, Retford, Notts, DN22 6BR.

Lettices

Where readers write! If you have a programming problem or can offer a solution to a problem mentioned here, write to the Lettice Column.

Mr. M. D. Cross (Membership #8887) writes:

I have been trying for some time to obtain a copy of Microsoft Pascal V3.2 from commercial sources. I would be grateful if any member can help with my quest!

Bryan Hopkins (Membership #8906) is also on a quest, this time for some information on graphics file formats:

I am interested in writing some software which requires the use of files created by various graphics packages, chiefly PC Paintbrush and GEM Draw.

So far, I have been unable to work out how the files these packages produce can be read to the

screen.

Possibly, someone else in the group has solved this problem already, and would be willing to pass the information on. Any assistance gratefully received.

Writing For C Vu

sub-titled "How to keep the editor happy"!

As you can see by flicking through any issue of C Vu, the group relies heavily on a small group of members who make the effort to contribute (which is, after all, what a user group is all about!). If every member of the group made the effort to write something for C Vu then we would have enough material for many issues to come.

Although articles are always welcome, within the group we have many members who can help with your technical queries - either about C itself or related to the operating system/environment you are using. At present we have members who can help out with technical queries on MS-DOS, OS-9, Unix, Xenix, Mirage, TOS (including GEM), Minix,....

Submissions for C Vu should be in the form of ASCII text files, on a 3 1/2" or 5 1/4" standard MS/PC/GEM-DOS format disk. 5 1/4" BBC Micro format (DFS or ADFS) single-sided/single-density (i.e. 100k) disks can also be accepted. The alternative is to upload articles to CIX, as detailed elsewhere in this issue.

Ideally, the only formatting in text files should be a double CR/LF between paragraphs, and between EACH line of source code (if using examples, etc.) Generally speaking, source code of great length will not be reproduced in C Vu, but will be placed in the Source Library.

Screen dumps are always welcome: IMG,GEM,PI (i.e. Degas), IFF, and MacPaint can all be accepted, but must be on DOS format disks! Lastly, please remember to include your name in any text files, otherwise we may be unable to include a "By" line!

Megamax Laser C
by Phil Stubbington

The Next Generation

Why Choose C?

C is NOT a beginners language, like Pascal or BASIC. C was developed to give the programmer the power of assembly language (well, almost) while retaining the readability of high level languages. Consequently, things like run-time error checking are entirely up to you. The disadvantages to this are obvious, but on the plus side is does make C very fast and versatile. A language like Pascal gives you no option when it comes to errors. Type in a number that is too large, and the program will stop with a message like "Integer overflow at line 90". In C, you could write some code that checked the number, and ask the user to re-enter the number if it was out of range.

One of the other advantages of C is that it is such a widely used language that finding a good reference book (including some specific to the Atari ST and GEM), or even a training course should be no problem. If your budget doesn't quite stretch that far, you will also find a vast quantity of C source code in the public domain, which is often of a very high quality and very useful for learning

about the language.

Overview

Megamax Laser C (MLC) comes on three single sided disks with a 640 page paperback manual, and costs £158 ($199 US). The System disk contains all the programs you need to get things running, although you will also need to copy the header files and run-time library from the Works disk. An addendum to the manual, and details of how to convert your existing Megamax code to Laser C are also included. The Works disk also contains the compiler controller (cc) and some example programs. The Utility disk contains 16 tools (utility programs) plus the debugging library and its' documentation.

After making backups of all the disks, installation is simply a matter of creating a directory called "MEGAMAX" in the root directory, and copying all the files and directories to it.

The Shell

MLC is the only C development system that comes with a proper GEM-based shell. Metacomco provide a very rudimentary GEM shell (Menu+) with Lattice C, but it doesn't do much to speed up the programming cycle. Other than that, you are stuck with text based shells most of which attempt to emulate the shells found on Unix systems (ie. cryptic commands usually limited to two characters and a total lack of integration).

Whatever size of program you are writing, most time is usually spent going round in circles – edit, compile, link, find bugs, edit, compile,.... and so on. As each of these stages would typically involve loading a separate program from disk, anything that can be done to speed up the process has got to be "a good thing". The Laser shell attacks this problem in three ways:-

- the editor is integrated into the shell,
- any of the programs (eg. compiler, linker) provided with MLC can be made RAM resident. This removes the time consuming transfer from disk (including RAM disks!) to memory, and
- frequently used files (typically the source and object code of your program) are held in a variable size RAM cache.

After the Laser shell is launched and the RAM resident tools are loaded, you are presented with the usual GEM menu bar and desk top.

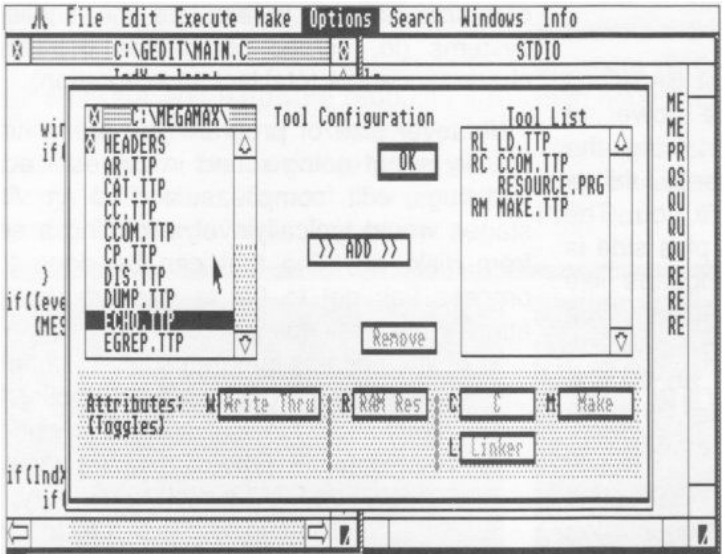
All the usual file management routines (copy, move, delete, rename, etc.) are built into the shell, and are available as separate tools. Unlike current versions of GEM Desktop (and most of the text based shells), copying files doesn't require endless disk swapping if you are stuck with a single drive. Strangely, the Laser shell has no routine for formatting disks, although the source code for such a program is included on the Works disk!

Once you have typed in your program, the next stage is to compile, link and run it. The entire process is carried out by topping the source window and pressing Control-R. Additionally, it will only compile those files where the source code has been altered since the last link. This facility is really a simplified version of the "make" tool described later on.

You can also specify which files are to be compiled and linked, the name of the final program, and if you want a symbol table generated (which you will when debugging). This information can be saved to disk if required.

Debugging programs is a pain at the best of times, but fortunately the Laser shell eases the process. When your program crashes control normally returns to the shell, and the STDIO (standard input/output) window will show exactly where you crashed. Additionally, pressing Control-Delete will stop any program in its' tracks (even GEM based programs, which otherwise would require a quick tap of the reset button).

There isn't really much you can say about RAM residency and RAM cache, apart from the fact that they work, and that they speed things up tremendously! It would have been nice though, if Megamax had given details of the file format for RAM resident tools, so that you could add your own.

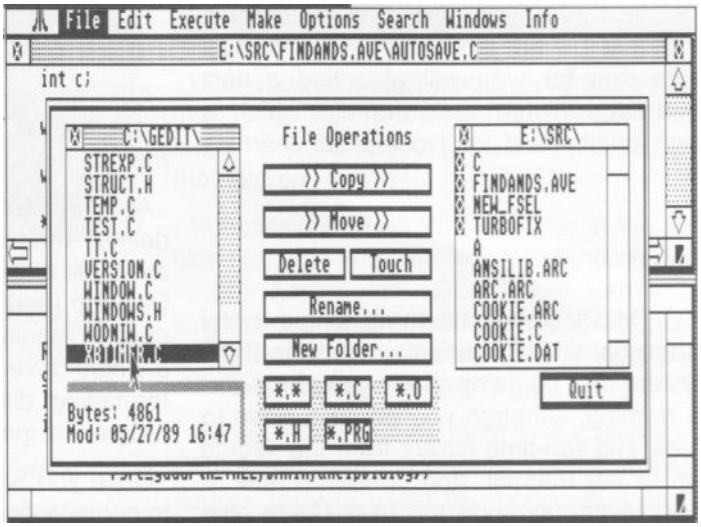


Any of the programs supplied with MLC can be made RAM resident.

Finally, the Laser shell allows you to run most text-based programs in the STDIO window, as long as they don't use any fancy control codes. I found this useful when using the "egrep" tool to generate a cross- reference list - I could simply cut the output from STDIO and paste it into the source code window. From STDIO you also have access to a very limited set of commands. For those that know it, the Laser shell operates in a similar fashion to the Macintosh Programmers' Workshop (MPW) - Megamax started off developing for the Macintosh.

The Editor

Although the editor is part of the shell, it is easier to treat it as a separate program. Up to five files can be kept in memory at one time, and you can cut, paste and copy between them to your hearts content! Standard GEM windows are used, although the scroll bars are a Megamax invention to allow very fast smooth scrolling (at a minimum of 10 lines per second).



All the usual file management routines are built into the shell. Unlike GEM Desktop, copying files doesn't require endless disk swapping with a single drive!

All the features of the original editor have been carried over to Laser, although you are no longer restricted to files of less than 32k, and control keys are handled more intelligently. The autosave option has been improved: you can now choose the maximum time between saves, and all "dirty" files are automatically saved before executing your program.

Nothing very remarkable has happened on

the search/replace functions - apart from some bugs being eliminated. You can search forwards and backwards, and replace once or always (with and without confirmation). The search string is taken literally - wildcards or anything more useful haven't been implemented. Strangely, the option to ignore case still appears as a separate menu bar entry, rather than being part of the search/replace dialog (the obvious place for it, I would have thought).

The Compiler

Megamax appear to have bought in the actual compiler from a company called Digital Lynx (no, I haven't heard of them either!) rather than improve their previous compiler. The compiler is still of the single-pass variety, and conforms to the original Kernighan and Ritchie (K & R) definition of the C language, with a few extensions (or, to put it another way, the compiler and library are not up to the forthcoming ANSI standard). There are no restrictions on program or function size.

Libraries

Two libraries are included with Laser C. The standard library contains what Megamax term "a large complement of Unix routines" as well as a full set of GEM functions. Routines are also provided to access the "Line-A" graphics primitives on the ST - useful if you want to use graphics, but don't want to do all the programming that GEM involves. Most routines have been coded in assembler for extra speed.

The second library is used in addition to the standard library, and is a "pseudo-debugger". Normally, debuggers come as separate programs, but the Megamax debugger actually becomes part of your program. Other than that, the library provides most of the facilities you would want from a debugger, and has the added advantage of being extendable.

Utilities

Around sixteen additional utilities are provided. Some duplicate functions built into the Laser shell. I shall briefly describe three of the major utilities, the first two started off life in the Unix world:-

- egrep - allows you to search for complicated patterns in text files. Generating cross-references is one application. Very useful for documenting.
- make - reads a makefile, which tells it how to rebuild a particular program. Really only of use if you have several source files, when it becomes essential. This is a full implementation, the Metacomco make isn't.
- rcp - resource construction program. Allows the easy creation of pull-down menus, dialog boxes, and icons. Has the dual purpose of being useful in prototyping, as well as making your programs language independent (any text, such as error messages, should be in the resource file, rather than the program itself). Essential for sane GEM programming, but this implementation is incomplete (ignores any alerts in a resource file). Buy K-Resource or WERCS instead.

Documentation

The main documentation for Laser C is a 640 page paperback, which is perfectly readable (and indexed) for a reference book. Additionally, there is an addendum and documentation for the debugging library on disk. The paperback contains a tutorial, details of the various programs and

functions provided, and several appendices.

Each function, or function group, starts on a new page, and starts with a brief synopsis of the function, and any parameters passed to it. This is followed by a paragraph or so of description, and a brief example or program extract. Strangely, although the actual library contains bindings for every GEM call, the manual doesn't document some of the less commonly used routines at all.

On The Bench

MLC certainly felt like the fastest C development system, but just in case my eyes were deceiving me, I carried out some simple tests. Using my Show GEM demonstration program (around 800 lines), I timed how long it took from invoking "cc" to seeing the opening alert box. With autosave off, and files in the cache, this took 18 seconds. Using Mark Williams' C (Version 3.0.5) the equivalent operation took 56 seconds (using the supplied RAM disk program "rdy"). Even with autosave on (so the source file would be saved before the compile started), the figure only increased to 42 seconds.

To get some idea of execution speed, I ran the Dhrystone benchmark (Version 1.1), which is generally regarded as a better indication of performance than the Sieve benchmark which is normally quoted. MLC gives a figure of 852 iterations per second, compared with 1149 for Mark Williams' and a dismal 650 for Lattice C. Higher figures denote better performance. Although I haven't had the opportunity to test it#yet, Manx Aztec C is likely to beat the lot.

Conclusion

As with any computer language, the developers have had to make a compromise between speed of compilation and execution. MLC is over 300% faster than Mark Williams' C on compilation, and 26% slower on execution. Compilation speed affects everybody, but the biggest difference in execution speed is brought about by using better algorithms, not by any optimisations the compiler does.

The Laser shell is a vast improvement. With the previous version, and with all the text based shells I have tried (including Mark Williams' "msh" and "gulam" - which is bundled with Aztec C) I tend to drop back to good old GEM Desktop to carry out simple operations like deleting and copying bunches of files, for speed. As I have already mentioned, the Laser shell is actually faster than GEM Desktop, and the editor is sheer bliss!

Documentation is also very good, with plenty of examples, which are often available on disk as well. An improvement over previous versions, and better than Mark Williams' C, apart from the few GEM routines that aren't documented!

Overall, MLC is a good buy. It is still the most compact C development system for the ST (it can be comfortably run from just floppies), and probably the fastest on compilation. Megamax have also mentioned real technical support and a newsletter - only time will tell!

Stop Press!

Since this review was written, Megamax have released Version 2.0 of MLC. The main difference seems to be support for a true source level debugger (Laser-DB). An update on MLC will hopefully appear in a future issue of C Vu.

Standard Watching
by Neil Martin

With Neil Martin of the British Standards Institute

Hello Folks!

Let me introduce myself. I am Neil Martin, a senior software engineer with the British Standards Institute. I am currently responsible for the setting up of a harmonised European Validation Service for C compilers and I am also a representative on the BSI C panel which is helping to produce the C standard. I hope to try and keep the members of the user group up to date on the issues.

The ANSI and ISO Standards

The whole business over the C standard is somewhat depressing. It was expected that the ANSI version would be ratified and published early this year, with the ISO version following about a year later. Unfortunately, there have been problems both at ANSI, and between ANSI and It seems unlikely now that a C standard of any form will see the light of day before September. There are also bad feelings in the UK at the lack of clarity in the standard - it is not impossible that we will end up with different ANSI and ISO C standards. If this happens, C users will, I guess, vote with their feet. The only consolation is that the final document is not likely to change much from the current draft. If you are interested, this can be obtained from the BSI Standards sales dept on 0908-221166. It is not particularly cheap at 30 pounds (approx.). However, the rationale, which is in itself a very useful document, is included. I believe you can obtain it cheaper direct from ANSI in the States if you can be bothered with the hassle.)

Special Interest C Groups

I would also like to try and encourage some of you folk out there to get involved in some of the more interesting specialist C groups. With this in mind, the following article gives details of a numerical extensions group recently formed in the USA. If anyone is interested in participating but does not have access to uucp (or JANET email) let me know and I may be able to arrange mail distribution from the UK.

Neil Martin British Standards Institute PO BOX 375 Milton Keynes MK14 6LL Phone 0908-220908 x2813 Email: neil@bsiqa.uucp Telecom gold 84:MNU174

The Numerical C Extensions Group
by Rex Jaeschke

Rex Jaeschke, NCEG Convenor, reports on the status of the group

Introduction

When I conjured up the idea for an ad hoc group to define numerical extensions to C earlier this year, I had no idea as to what the reaction would be. The evidence is now clear that this endeavour is seen as being very worthwhile. Not only have more than 90 people asked to be added to the contact database, but 30 of them attended the one-and-a-half day meeting at Cray Research on May 10 11. The backgrounds of the attendees was diverse. The supercomputing industry was represented via Cray, Convex, Supercomputer Systems, and Thinking Machines.

The IEEE community was well represented by Hough (from Sun). Cody (from Argonne Labs), and Thomas (from Apple.) Other organizations represented included Unisys, Microsoft, Digital Equipment Corporation, H-P, CDC. IBM. Solborne, Farance, Inc., University of Minnesota, Inter-metrics, and Information and Graphics Systems. The digital signal processing industry was represented by Analog Devices, and LLNL, Army BRL, and Polaroid Corporation represented the user community. Dennis Ritchie from AT&T also participated. There was no real sentiment that we deliberately go against the direction established by ANSI C. In fact, quite the contrary. However, it was recognised that some of ANSI C s constraints may impede our activities resulting in possible conflicts. The whole issue of errno and formatted I/O of NaNs and infinity are examples.

Main Numerical Issues	
aliasing	29
vectorization	27
complex	27
variably dim arrays	25
IEEE issues	24
exceptions/errno	24
float/long double library	23
parallelization	22
ANSI <math.h>	21
array syntax	19
extra math functions	17
aggregate initializers	15
inter-language issues	15
wide accumulators	10
math function precision	9
non-zero-based arrays	8
numerical representation	6
new data types	4
new operators	4
function overloading	4

The Issues

The main purpose of the meeting was to identify and prioritise the principal technical issues. The group then voted on each topic indicating high or medium (or no) priority. The high priority votes were weighed twice as much as the medium, and a list of priorities resulted (inset).

Another topic, "Arrays as first class objects had a high priority (21) but after considerable debate was dropped from the list since it was agreed its addition would likely cause great confusion to existing C programmers. " Formation of Subgroups The bulk of the agenda time was then given to the top ten topics, each getting 20 30 minutes. For each of these topics, attendees volunteered to be the primary and alternate co-ordinator. (The minutes of the first meeting identify these people. In the interim,contact me for details.) The intent is that the real technical work will go on between meetings and be coordinated by the leaders of each subgroup. Then, at the following meeting, each subgroup will present the results of its work and make formal proposals as appropriate. This way, the committee can focus on the final, distilled issues rather than everyone getting involved at all levels. It will also significantly reduce the amount of paper in the mailings. If you wish to participate in any of these subgroups it is your responsibility to contact the leaders and identify yourself, your concerns and how you can help. If your area of interest is not listed here, start your own subgroup and let me know. Mailings and Submissions Most of people interested in NCEG appear to have an e-mail address so that should

make the subgroups job much easier in coordinating various viewpoints and proposals. However, all formal distributions will be by paper mail. Since meetings are to be once every six months there will be two mailings between meetings. The first will occur within 4-6 weeks after a meeting and will contain minutes, new papers and other appropriate correspondence. The second will occur about 4 6 weeks prior to the following meeting. The cut-off date for formal submissions for the September meeting is August 11. Forward all correspondence to me (either by mail or via uunet!aussie!rex) and I will assign it a document number. (Note that I do not have a troff formatter.) However, do that only if your paper is concerned with issues other than those being handled by the subgroups. For subgroup issues, forward papers to the subgroup coordinators so they can include it in their submissions to me. The intent is to avoid excessive duplication of points and to allow the short meeting time to be used more effectively. The more formal documents we have the slower it will go. Tom MacDonald at Cray Research has agreed to do the mailings, at least for the interim. Frank Farance of Farance, Inc., has volunteered to be the redactor of the group s working document. Thanks to Tom and Frank. (Thanks also to Randy Meyers from DEC, who acted as meeting secretary and to Cray for being meeting host.) Formal Affiliation There was general consensus that we become affiliated with a recognized standards organization. The final proposal was that we become a working group within X3J11. If we follow that route, it will result in our publishing a Technical Report, a non-binding report on our findings and recommendations. With suitable planning, we might be able to have that elevated to a Technical Bulletin and get it distributed with the ANSI Standard. Getting our extensions adopted as a standard is also possible, in the long term. At this stage, I plan to ask for agenda time at the next X3J11 meeting to discuss admitting us as a work group.

In the interest of economy, the next two meetings are scheduled in the same location and week as those of ANSI Cs X3J11. These NCEG meeting dates are September 19 20 (Salt Lake City, Utah), and March 7 8, 1990 (New York City.)

Let me know if you want a sample issue of the Journal of C Language Translation. If so, provide your mailing address.

Contacts

If you would like to get in touch with Rex Jaeschke he can be contacted at The Journal Of C Language Translation, 1810 Michael Faraday Drive, Suite 101, Reston, Virginia 22090, USA. For any readers on the other side of the pond, or with limitless pockets telephone (703) 860-0091.

Copyright & Things

C Vu is composed entirely on an Atari 1040STF, using an Atari SH205, and the disk version is created using just about any and every text editor you care to mention - STedi, MicroEMACS, ST Writer Elite, Turd Perfect, etc., etc.

ISSUE 6 DEADLINE

All material for issue 5 should be submitted by 1st September 1989. Any material submitted after this date is unlikely to appear in Issue 6.

Please, please, please include a draft printed copy of any articles along with the text file on disk. Remember to include your name, title and/or curriculum vitae (!) in the article.

If you have any queries about the group, contact:

Martin Houston at 36 Whetstone Close, Farquhar Road, Birmingham, B15 2QN
Telephone: 021-454-3448

Alternatively leave a message in the c_users_group conference on CIX or mail to mwhouston pr philip.