Volume 1 Issue 4

C Vu

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

Mail: 36 Whetstone Close, Farquhar Road, Birmingham, B15 2QN

Telephone: 021-454-3448

Contents

Editors Note:

Apologies to those authors who have not been given a byline in this issue. To prevent this from happening again, please remember to include your name, and membership number actually in the text of the article itself, and in any source code extracts if these are separate.

Phil Stubbington

Three Transputer C's
by Dr Howard Oakley

Rather than just reviewing three C compilers for the Transputer, Howard Oakley examines their different approaches to introducing parallelism into the language.

The Transputer is a remarkable chip. Founded in part on the theoretical work of CAR Hoare on Communicating Sequential Processes (CSPs), it has microcode which allows it to timeslice automatically through whatever processes are ready to run on it, offering internal pseudo-concurrency. It also provides simple 'channel' communications between such processes. Such a very fast (10 million instructions per second) processor is already quite a clever beast, but if you then add high-speed communications links, to enable you to build arrays of processors which can pass messages at 20 megabits per second, you have a winner – if you can program it.

The need for processors like the Transputer was established many years ago. No sooner had the first real electronic computers been built, than people started predicting theoretical limits to the ability of a single processor's speed. It was clear that the way for the future was to process in parallel, a technique which had reached its peak in the 1930s, when rooms full of clerks with mechanical calculators did the same thing to try to solve the more complex mathematical problems of the day.

By this stage, those who have followed the development of the C language will have guessed that I am about to discuss three implementations of Concurrent C, as described by Gehani and Roome (Software: Practice and Experience vol. 16 no. 9, pp. 821-844, 1986). I am not. Concurrent C,

although based on Hoare's CSPs, was developed at AT&T Bell Laboratories on a Unix system, and shows clearly its origins. Processes have to be created explicitly and in detail; communications are known as transactions, and can be selected and accepted in elaborate code. In short, Gehani and Roome opted for language extensions rather than library functions, and an approach not unlike that of Ada, that expansive language which appears to have had little impact on life except where it has become politically mandatory! Concurrent C seems to have slipped into relative obscurity as a result.

The Transputer poses rather different problems. In the first place, its time-slicing is transparent and essentially uncontrollable. It has no facility for the robust management of memory, only one operating system to speak of (Helios, developed by Perihelion Software, a sympathetic interpretation of Unix which comes with its own C compiler of excellent pedigree - the Norcroft one of Archimedes fame - which is beyond the scope of this article), and a systems programming language (Occam) which is extremely sparse and un-C-like. The Transputer also lacks interrupts, and can only run processes at two different priorities (the higher priority even being sans timeslicing). All this is very different, and has resulted in three rather disparate solutions to the implementation of C with parallelism.

OCCAM - THE MODEL

Occam is like a racehorse - very fast and ideally suited to what it is intended, in its case

programming the Transputer in parallel, but pretty useless if you want it to tow a cart (such as when writing large applications programs). Creating parallel processes is just a matter of three letters and a bit of indentation, thus:

```
PAR
SEQ
    -- this is sequential process one
SEQ
    -- this is sequential process two, in parallel with process one
```

Communications by means of channels is also very simple,

```
CHAN OF INT one.to.two:    -- declare a channel of integers (32 bit)
PAR
INT digit:
SEQ
    digit := 1             -- assignment
    one.to.two ! digit     -- output digit through the channel
INT received:
SEQ
    one.to.two ? received  -- input received from the channel
```

There are several other innovative constructs in the language, such as the alternative ALT, which waits for the first of one of a series of alternatives to become true, and a multiple-choice IF. Many of these can be replicated (the Occam for for() being SEQ i=0 FOR n, for instance), and some prioritised (PRI ALT and PRI PAR).

However, the cost of these is the lack of data structures (although these can be imitated with common indices into different arrays), no dynamic allocation or pointers (again, ingenious tricks can mimic them), and what is essentially a very static language overall. This is hardly what will tempt today's C programmers to the Transputer.

3L'S PARALLEL C - THE STANDARD

When Inmos, purveyors of Transputers to the world, first wanted Fortran, C and Pascal compilers for their systems, they invited Lattice Logic Limited of Edinburgh to provide them. Unfortunately, Inmos have always considered such languages to be inferior to Occam, even terming them "Alien Languages" in public. The team from Edinburgh separated from their parent company last year, forming 3L Limited primarily to develop and market a range of parallelised compilers, of which Parallel C and Parallel Fortran are already available. Consequently, the front ends of their compilers and much of the code-generation are well-proven and thoroughly debugged.

Parallel C does exactly the opposite of Concurrent C, and implements parallelism primarily by means of library functions. However, 3L have been careful to provide for great flexibility in just how the programmer uses such features. Two basic models are offered: the simpler assumes that the entire application will consist of just two code modules, a master (which makes all the operating system calls to the host, and resides on the Transputer which is connected to it) and an indeterminate number of workers (which can only communicate with the master). This is ideal for a 'farm' hardware model, which considers a Transputer array to consist of a root and any number of workers which will run identical code. The more general model allows for any number of quite different code modules, which the programmer will carefully distribute onto the hardware, taking into account each Transputer's connections, etc.

These are best illustrated in terms of their typical uses. The 'farm' model, using 'flood filling' of processors within the farm, is ideal for calculating the Mandelbrot set: a problem which requires the independent calculation of many thousands of point values. In this, an example program provided with Parallel C, the master code decides how to divide the calculation up, and then sends out to each worker module the information it requires (the top left co-ordinate, length and interval for each module to work from, for example). Each worker then receives this information by courtesy of a code module provided (called the router), and carries out the mass of calculations. When finished, it returns the values to the master, which displays them on the screen.

The 'general-purpose' approach is exemplified by a program which compacts and encrypts a file using a linear pipeline of Transputers. In this case, the best algorithmic approach is to pour a stream into one end, get each processor to carry out part of the sequence of operations required, and then save the resulting stream from the far end of the pipeline. In this case, each Transputer has very different code modules, and a specific role according to its position. If we are using five processors, there will be three separately compiled and linked modules to be loaded onto the pipeline, using the general-purpose distributing loader, and modules concerned with reading in the raw file, and writing out the processed one, for the two ends.

The development of programs thus requires an additional final step, that of configuration. Whether you are flood-filling or using the general-purpose approach, each code module is written separately, compiled, and then linked. When all the modules are complete, they are then bundled up together by means of a configurer (two varieties of which are supplied - a flood-fill configurer, and a general-purpose one). The configurer requires a text file describing the relation between the software and the hardware - a configuration file - which is very simple and non-specific in the case of flood-filling (it copes with any number of Transputers at run-time), but an exhaustive and detailed account for the general-purpose approach (which is then tailored to that, and will only run on that, hardware set-up).

We have so far only talked of the coarser level of parallelism offered by Parallel C. Occam-style PARs, time-sliced on a single processor, are implemented as threads, library functions which are very simple to use:

```
/* a multiplexer implemented in 3L's Parallel C */
#include <chan.h>                  /* headers for channels, threads */
#include <thread.h>                /* and semaphores                */
#include <sema.h>
char buf[1024];
SEMA buf_free;          /* semaphore to control global access to buf */
CHAN **in_p, **out_p;           /* pointers to port/channal vectors */
main(argc, argv, envp, in_ports, ins, out_ports, outs)
int argc, ins, outs; /* note all the extra parameters for channels */
char *argv[], *envp[];
CHAN *in_ports, *out_ports;
{
    extern void receive();
    int i;
    sema_init( &buf_free, 1 ); /* init semaphore to say buf is free */
    in_p = in_ports;       /* juggle ports/channels to make them OK */
    out_p = out_ports;
    for (i=0; i < ins; i++ )    /* for one thread per input port, */
        thread_create( receive, 50*sizeof(int), 1, i );
           /*   create thread using receive(), with workspace size */
           /*  50*sizeof(int), 1 argument, and using port/channel i */
}
void receive(i) /* the process to handle messages from single input */
```

```
int i;
{
    int msglen;         /* each thread uses its own message length */
    for(;;) {
        chan_in_word(&msglen, in_p[i]);     /* await next message */
        sema_wait(&buf_free);   /* wait until access to buf is free */
            chan_in_message( msglen, &buf[0], in_p[i] );
                /* read body of message into buf from local channel */
            chan_out_word( msglen, out_p[0] );
                    /* send message length to output port/channel */
            chan_out_message( msglen, &buf[0], out_p[0] );
                    /* then send the message to output port/channel */
        sem_signal(&buf_free);     /* then release semaphore on buf */
    }
}
```

As far as the rest of Parallel C is concerned, it is very standard Kernighan & Ritchie, getting close to Draft ANSI, with the only notable exception being the continuing omission of the enum type. Host facilities are accessed via the standard Inmos file-server, which though not fast is standard and well-proven. 3L now provide an additional graphics library which makes PC graphics much easier, too. Because 3L still use the Inmos linker, slow though it is, it is possible to link together (and configure) code from their Fortran and Pascal compilers, as well as assembler and Occam. Although the Parallel C compiler does not compile to intermediate assembler code, in-line assembler is fully supported.

LOGICAL SYSTEMS' TRANSPUTER TOOLSET - A CROSS-COMPILER

In the UK, it is very easy to assume that, because Inmos designed and manufacture the Transputer, and Inmos are very British, our US friends lag behind when it comes to using it. This is a dangerously myopic view, and one which could prove disastrous when it comes to future success. Logical Systems of Oregon offer a full-featured set of cross-compilers for Transputer targets, with the ubiquitous PC, Macintosh, and other computers as hosts. In order to encourage users of other hosts, the product is also available (at about double its basic price) with full source code – almost worth the money just to study a fine compiler implementation in source!

The approach taken to software configuration is quite different from that in Parallel C. Logical Systems supply a general-purpose network loader LD-NET which uses the general-purpose style, but does not require that configuration directly follows linkage. Development is by a conventional compile-assemble-link sequence, to generate individually whatever program modules are needed in the complete application. These remain separate executable files, and LD-NET is called to start the program. It reads a 'network information file' (far simpler than 3L's configuration file format) and then broadcasts the separate executable files onto the array of Transputers. Each Transputer in turn retains the file(s) which it has to run, and sends on copies of all the code files.

This makes the Transputer Toolset easier to use for applications development in many ways. If you write a program which is designed for 12 Transputers, say, then both Parallel C and the Transputer Toolset will allow you to get the most out of them - if that is what the user has. If you use the Parallel C flood-fill method, then someone else with just one Transputer will still be OK; on the other hand, many programs are not amenable to that approach, and you then discover that the user has to reconfigure if not rewrite your application before it can run on his single Transputer. The Transputer Toolset gets around this by making the decision as late as possible, i.e. at load time, so all the user will have to do is re-edit the network information file in order to cope with his lesser

resources. The cost of this is inefficient loading - LD-NET can take a long time to load a network, and, as each Transputer must handle every code file, you do generally need plenty of memory on each.

Logical Systems have also chosen to implement parallelism in terms of library functions, arguing that this enhances portability and prospects for the future. Two complete sets of functions for parallel programming are offered: the first, based on a paper by Jeffrey Mock (of Pixar, another US firm to watch over the coming years), essentially provides library functions for all the important Occam features, such as

```
ProcPar(p1, p2, ... pn, 0)      /* is Occam PAR    */
   Process *p1, *p2, ... *pn;
ChanOut(c, cp, cnt)          /* is Occam c ! [cp FROM 0 FOR cnt] */
   Channel *c; char *cp; int cnt;
ProcAlt(c1, c2, ... cn, 0)      /* is Occam ALT    */
   Channel *c1, *c2, ... *cn;
```

If you come from a Unix background, though, you may prefer the alternative fork()/join() model, illustrated in this next skeleton example:

```
#include <conc.h>                    /* the concurrency header file */
 ...
 int (*tsk2)();          /* pointers to 2nd and 3rd processes */
 int (*tsk3)();
 char ws2[1000];         /* workspaces for 2nd and 3rd processes */
 char ws3[800];
 Forkblk f;                          /* fork status information */
 PDes p2;            /* process descriptors for 2nd and 3rd p */
 PDes p3;
 int a;                  /* example parameter for 2nd process */
     PForkInit(f,3);          /* initialise f for three processes */
     p2 = PSetup(ws2, tsk2, 1000, 1, a);
                /* set up process 2, with a as the only parameter */
     p3 = PSetup(ws3, tsk3, 800, 0);
                         /* set up process 3, with no parameters */
     PFork(f,p2);            /* start process 2 at current priority */
     PForkHigh(f,p3);         /* start process 3 at high priority */
     /* during which 2nd and 3rd processes also PJoin() to finish */
     PJoin(&f);                  /* current process is done too */
```

Both the Mock model and the fork()/join() one only operate at the level of threads, using 3L's terminology, or local PARs in Occam. So, just as with Parallel C, you are forced to commit processes to being either distributable onto separate processors, or forever doomed to be mere threads, when you write your code. This has serious implications when we consider how users are going to want to use Transputers. If we are to realise the full potential of these chips, we are going to say that, if it does not run fast enough with n Transputers, then throw some more in and it will run faster. It will, if our code can make use of them - and there's the rub. With either Parallel C or the Transputer Toolset, there is always a programmer-imposed upper limit as to how many Transputers can be gainfully employed on any program; the sole exception to this is with the flood-fill technique in Parallel C.

The other details of the Transputer Toolset are consistent with a high-quality product: it is now almost completely up to Draft ANSI standard, although it still lacks some floating point library functions for the T414 model Transputers (they lack the arithmetic co-processor found on the T800 model), and bitfields remain unimplemented. The documentation is at times very skimpy, but

enhancements are promised. When it comes to performance, the Logical Systems linker is far faster than the Inmos one, but this makes it incompatible with all other Transputer languages, including 3L Fortran and Pascal, and Inmos Occam. Because the compiler uses assembler as its intermediate code product, use of mixed C and assembler is very easy. Finally, the Transputer Toolset is structured as a traditional C compiler, with the Pre-Processor separated out too.

DEFINICON'S TCC - COMBINING OCCAM AND C

The third C compiler is also an American product, from Definicon Systems of California, who are specialists in products which 'make PCs (and others) go faster'. When they were deciding how to implement parallelism in C, they consulted the likes of 3L, and received the advice to do so using function libraries. Fortunately, they did not, and have produced what is surely the most unusual of the three compilers. As the principal designer of the TCC compiler, John Poplett, has already written at length in Byte (February 1988, pp. 249 - 254) and in .EXE (August 1988, vol. 3 issue 3, pp. 34 - 39), I will try not be too lavish with example code.

The TCC compiler takes heed of Concurrent C, and implements parallelism primarily by providing language extensions. It is possible to use the compiler in purely sequential mode, in which case a different run-time library is used, and all attempts at parallelism are thrown to the wind. The manual actually recommends that you restrict yourself to this initially, until "TCC's sequential and parallel compilation modes are thoroughly understood", indicating that further understanding of the resulting assembler output is really a pre-requisite to compilation in parallel mode. Daunted by this (and the fact that even in Inmos assembler is only used when it really cannot be achieved any other way), and the fact that I only have about 18 months experience of serious C and Occam programming, I have tried a little parallel work.

Superficially, the parallel constructs fit easily into C. For example, an Occam PAR would become

```
par {
        process1();
        process2();
        ...
        processn();
}
```

and ALT works very similarly. The snag is that when you want to implement programs for real, and on multiple Transputer systems, this simplicity gives way to a rash of #pragmas. Take the following program, which runs the Hello World! demonstration on a two-Transputer array:

```
#pragma transp 1                /* this is for Transputer number 1 */
#include <stdio.h>
#include <stddefs.h> /* note no messy parallel header files here! */
#pragma links L_HOST 2 L_NC L_NC     /* describes link connections */
#define LINK1IN (channel *)0x80000014           /* locate link 1 */
main()
{
    for(;;)
        putchar((char) *LINK1IN);    /* casts to char for link I/O */
}

#pragma transp 2                /* this is for Transputer number 2 */
#include <stddefs.h>
#pragma links 1 L_NC L_NC L_NC       /* describes link connections */
#define LINK0OUT (channel *)0x80000000          /* locate link 0 */
main()
```

```
{
    static char *msg = "Hello World!\n";     /* is this familiar? */
    char *ptr = msg;
    while(*ptr)
        *LINK0OUT = *ptr++;          /* output message over link 0  */
}
```

I suppose that this is roughly equivalent to incorporating the Parallel C configuration file as #pragmas into the source code, and then using a general-purpose configurer to fix it all to the hardware. More sophisticated pieces of code rapidly lose my concentration altogether, and I fear must border on the unreadable and unmaintainable. Perhaps this was the reason for 3L and Logical Systems abandoning the language extensions advocated in Concurrent C?

I must also point out that the TCC C compiler is an immature product as yet. To date, I have received two versions, neither of which was capable of running the very simple Dhrystone benchmark demonstration as supplied with them, although once I had stripped out all the #pragmas which I could not immediately understand, and converted the source back into ordinary C, they compiled and ran quite happily. When it has had time to settle, I expect the TCC compiler to be very close to Draft ANSI standard, and it works as a traditional C compiler in producing intermediate assembler code, thus supporting the use of assembler too. Also provided is albeit a somewhat rudimentary debugger - a key feature which 3L and Logical Systems are working on very hard - which can probe the memory of a Transputer from an adjacent one in the array. The latest linker is claimed to be compatible with the Inmos ones, and thus to allow linkage of other compiled languages, but I have not yet had the courage to test this.

WHICH IS BEST?

You will have gathered by now that none of these three C compilers really comes up to what I would like. But then I, like everyone else trying to program Transputers, would like the impossible - C made parallel with no extra effort required of the programmer.

I have deliberately avoided quoting performance benchmarks until now, as they are almost unhelpful. The Transputer has a certain amount (2 Kbyte for the T414, 4 Kbyte for the T800) of 50 ns RAM on the chip itself, and the way in which you use that, and how you run the necessary file server to communicate with the host, are critical. With a bit of juggling, choice of the right benchmark, and so on, the worst of compilers can appear to generate Formula 1 code. So, to equally stunt each of the three, I took the Dhrystone 1.1 benchmark, used no tricks at all, and recorded these results on my MicroWay MonoPuter:

HOST	COMPILER	Dhrystones/sec	Size (bytes)
IBM PC XT, 4.77 MHz 8088	Manx Aztec C 4.10	404	5898
IBM PC XT + T800-20	TCC 2.0 & 3.05	2451	15143
IBM PC XT + T800-20	Transputer T'set 88.2	4179	14506
IBM PC XT + T800-20	3L Parallel C 2.0	5213	40335

to which you might compare a VAX 11/780 at 1500 Dhrystones/sec. This puts a single T800-20 at about the same speed as a VAX-784, and considerably faster than a 16 MHz 80386 machine. The

Norcroft C compiler running under Helios on a T800-20 returns between 3500 and 4000 Dhrystones/sec, demonstrating how little overhead Helios imposes. The picture on floating point benchmarks was about the same, giving a best performance (by the Transputer Toolset and Parallel C) of about 1.3 MegaFLOPS. Claims by Definicon are that, using maximum trickery, they can squeeze over 7000 Dhrystones per second out of a T800-20 are matched by similar figures from Logical Systems and 3L, so it would appear that their best performances are very similar.

One final word of caution: be prepared to grope round the back of your PC if you are using multiple Transputers in it. Different compilers take different approaches to resetting and thus rebooting Transputers downstream of the root one, and you may need to change around the connections to accommodate each compiler's desires. Parallel C uses the 'down' reset, analyse and error lines, whilst the Transputer Toolset and TCC use the 'subsystem' ones (as recommended by Inmos). The sooner that hardware and software manufacturers standardise on this, the easier life will become. Furthermore, Inmos, 3L and Definicon each provide 'worm' programs to explore Transputer networks, none of which works correctly on my array of six T800s, although programs can!

After several months of playing intermittently with these products, I must confess that I am happy to have all three. Whilst Parallel C is by far the most polished and professional, so it ought to be for about £750. The Transputer Toolset generates far smaller executable files, and its approach to configuration is nearer that needed in a commercial product, but you get very little documentation with it for £295 (without source code). Definicon's TCC generated a great deal of interest at first, but I think that it is a little too complicated for me personally to justify its price, £495. Until TCC is more mature, I would recommend Parallel C if you can afford it, the Transputer Toolset if you cannot, and both if you really want to invest in a Transputer future. Happy programming!

3L Ltd are on 0506 415959; Logical Systems are on (USA) 503 753-9051;
Definicon Systems (UK) are on 01-498 0704; Parallel C and Transputer
Toolset compilers are distributed in Europe by MicroWay (Europe) Ltd. On 01-541 5466, the TCC compiler by Definicon.

Are You A Nutter?

Acorn Machine Owners now have their own sub-librarian:

Dr A. Gwyn Williams has offered to become a C User Group sub-librarian for the Acorn Archimedies. He writes:

> "I would be willing to act as a sub librarian for the source code library. I have an Archimedes with PC compatibility, and can transfer files to Acorn's ADFS format discs, readable on the Master, Master Compact or Archimedes; they are also readable by a BBC with the 1770 disc upgrade and ADFS ROM; I can produce 3 1/2" and 5 1/4" discs."

If you own an Acorn machine and wish to obtain CUG library material in 'native' format you can contact Dr Gwyn Williams at:

69 Boyton Road,
Hornsey,
London N8 7AE

Hertfordshire Local Group

Mr David Scott would like to hear from any other group members in the Hertfordshire area with a view to a local meeting on the 'C' Language and related topics.

If you are interested David can be contacted at

1 Park Close,
Markyate,
Nr St Albans,
Herts AL3 8RG

DON'T LIVE IN HERTS? THEN WHY NOT ORGANISE IN YOUR AREA. Just drop me a line for publicity in the next issue of CVU. I would appreciate it if all local organisers keep in touch with me so that everyone can be kept informed of any good things that are going on!

QL Owners' Can Now Access The Source Library

With a low cost PC-to-QL translation program

QL users can now translate and compile programs from the CUG library.

A program has been published by David Walker (22 Kimpton's Mead, Potters Bar, Herts. Tel. (0707) 52791) which will translate the library discs into QL format. It costs £12 and can be provided on microdrive cartridge or on disc. It is called IBMCOPY.

The program is set up to expect a two disc system, but the devices can be changed by the user.On a single disc system, the library disc is placed in flp1_ and a formatted microdrive cartridge is placed in mdv2_ to collect the translation. The program cartridge can then be placed in the remaining microdrive housing, mdv1_, and the program is started with the command 'exec_w mdv1_IBMCOPY'.

The program first provides the user with a directory of the IBM disc.The information provided includes the length of each file in bytes, and sub-directories can be handled.Selection of the file to be translated is made by cursor, and translation initiated by pressing the ENTER key.The bytes are metered as they pass over, and the byte count is displayed(c.f.the action of a petrol pump).

I have used files prepared in this way as source code for the Metacomco Lattice compiler.The text editor reads them in readily enough, although there is a superfluous byte in each line, just before the newline (a CR byte).Fortunately, these do not appear to affect the action of the compiler.This compiler is fastidious about types, and it generates warning messages over and over again in some files (e.g. CFLOW_C on disc 1 in which unsigned character pointers are confused with character pointers). Warning 30, 'pointers do not point to same object', is now engraved on my innards. The compilercan however bebamboozled; integers and FILE pointers are mixed up in the program 'ABOUT_C',but it does not spot the error.

A disc in IBM format is required to make the reverse translation, QL to IBM, because IBMCOPY cannot be used to format. I used one of the CUG(UK) library discs for this file.The file was written using the Metacomco text editor, and subsequently filtered to add the CR bytes mentioned above before translation.

The release of IBMCOPY sent to me late in April was not a final production version. The translation table permitting byte substitutions during the translation process was not working, and

there were problems with the deletions of files on the IBM disc. I understand that thesedifficulties have now been eliminated. Nevertheless, the program is well-produced and effective, and it represents a major breach in theQL-PC compatibility barrier. Full use of the CUG library therefore seems to be perfectly feasible for QL users.

Interested in GEM?

Especially ST owners. Then get in touch with other members interested in writing applications

Mr A. J. Wimble (member 8852) writes:

> You mentioned that CUG would use the newsletter to put people with similar programming interests in touch. Well I am interested in contacting anyone writing GEM applications, particularly on the ST, with a view to exchanging hints, information, routines etc.

If you are interested in contacting Mr Wimble drop me a line and I will put you in touch.

Minix On The Atari ST - First Impressions By Jwahar R. Bammi

*Direct from UseNet, possibly the first review of Tanenbaum's Un*x look-a-like*

Distribution

Publisher: Prentice Hall
Order #: 0-13-584391
Manual ISBN: 0-13-584434-7
Price (US) : $79.95 + shipping
Authors : Andrew Tanenbaum Johan Stevenson Jost Muller

9 single sided 3 1/2" floppies + 70 page manual. The OS book by Andrew Tanenbaum is not included in the above package. If you are knowledgable about Unix, the book is not required to run/ use Minix, but in my opinion even if you are, the book is still very worth buying. Highly recommended reading.

Hardware Required

Minix will run on almost any ST/MegaST configuration. Single/Double sided floppies, hard disk (you need a atari sh204/212 hard disk, more on this later). The console driver supports both mono (25 and 50 lines) and color monitors. There is support for various national keyboard types (via an utility). As distributed, it is configured for a minimal ST (1 single sided drive, 512K memory, any monitor). There is adequate documentation in the manual on how to reconfigure for the various hardware configurations. The built in clock on the megaSt is supported.

Software Supplied

Comes with complete source and binaries as described in the Minix book. Also included is a C compiler, libraries etc. The source for the C compiler is not included, the library source is all there. Left as an excercise for the reader are drivers for the serial port, midi, and cart. ports. The distribution includes drivers for floppy, hard disk, keyboard, screen and printer, all with source. I have begun writing a serial driver, and it looks like its going to be do-able.

Installation

The instructions in the manual are more than adequate to install Minix on any configuration. The installation went very smoothly on all the configurations i tried. One small point in the installation instructions. The instructions seem to use the word `sector' to mean `block' when referring to the third parameter of `mkfs' in the instructions for setting up hard disks. The useage is correct for floppies, but not hard disks in my mind. This caused me a little grief because i was not thinking and blindly following the instructions, but no big deal.

Experience

For some of us old hands from the PDP V6/V7 era, this is almost like deja vu. It smells like, it feels like a Pdp v6/v7 unix system but with the additional benefit of having a large linear address space thrown in. Minix will use as much memory as you have. The root file system is loaded into ram disk (which thereafter becomes the root partition) from floppy/hard disk on startup. The size of the ram disk is determined by the size of the partition it is initialized from, so its totally flexible. The minimum size is around 140K, a very workable size i have found to be 256K, and if you have the luxury of globs of memory, you can make it as large as you want.

The hard disk support allows you to use any partition(s) on any drive(s) for minix file system(s), so it co-exists with TOS partitions quite nicely. The hard disk driver supports multiple drives too. Utilities are provided to read/write/list directories from/to TOS partitions on hard disks or a floppy. The differences between the IBM Pc and ST versions are adequately documented. There is an interesting chapter in the manual about the ST implementation, especially the bit about implementing fork()/exec() semantics on a 68K.

There is also adequate documentation in the manual for recompiling Minix itself on various hardware configurations and also on recompiling the stuff using the Alcyon compiler. The only place where the documentation is really weak is on details of the C compiler, libraries and associated utilities. Except for the C compiler details, i will not complain too strongly because the source for the libraries and associated utilities are in the distribution, and i have a unix system right next to the St to do a man anytime I want.

The C compiler seems to be quite stable, though the code produced is nothing to write home about. My biggest gripe would be that it uses 6 passes to produce an executable (cpp/parser, optimizer, code generator, assembler, linker and load format converter). An interesting project would be to combine at least the linker and the converter, and maybe the optimizer and the code gen. Unfortunately the sources for these are not supplied. Performance wise, Minix is no speed daemon, nor is it intended to be. But the source is "with you", and that leads to some interesting possibilities... (i wish i didn't have a thesis to do :-)

More experience

In my way of looking at things, to really get some experience, you got to roll up the sleeves and get your fingers dirty. So I set out to port MicroGnuEmacs (mg V2a). I am happy to report that it is up and running under Minix, with all the facilities like Dired, regex, forking shell etc etc. I also went into the Minix archives at bugs.nosc.mil and pulled down some upgrades and utilities.

So far I have the following:

df/mount/umount	enhanced to store the state in /etc/mtab, df reports bsd style sizes in K bytes or the traditional i-node style
ed	I had to have this to start me up, mined was driving me up the wall. Now that mg2a is up i don'nt really need it.
ls	enhanced ls, does columns, puts * and /'s etc
make	pd make (from the same original source as the TOS make that we posted the diffs for). The supplied Make is a little strange.
mg	MicroGnu emacs V2a
more	enhanced more (pager)
sh	the sh as distributed by ast for v1.3 of minix

Be my guest and grab any of this stuff via anonymous FTP from dsrgsun.ces.cwru.edu (129.22.16.2) from the directory ~ftp/pub/minix. A note about the Mg port: I haven't worked in the key binding stuff for special keys (function, arrows) as yet, but i will be doing so. So you may or may not want to grab Mg just yet.

More on Hard Disks

The supplied hard disk driver works fine with an atari SH204/212 hard disk and the atari supplied HDX. I also tried with a Supra 20M. It almost works, "close but no cigar". It gets some DMA interrupt which it doesn't like, and it even manages to do a mkfs, copy files etc, but you keep getting spurious interrupt messages, and finally the kernel panics - el crashola. So quite obviously only a small fix is need somewhere in the hard disk driver code, and since Supra also supplies the source to its Dma functions, it seems like it should be do-able to put 2 and 2 together.

A few random notes:

- always #define ATARI_ST when including stuff from the system include directory, otherwise it leads to errors.
- use chmem on cem and cv for even medium size progs.
- the document suggests that you create a 512K ram disk on a 1 Meg ST. This is nice because you have a large /tmp and also have space to shove a few passes on the C compiler into /lib, but it creates problems when you have say mg running from a shell, and you try to fork(). The fork() fails always due to lack of memory. I set up my root in a 256K ram disk, that gives me enough space in /tmp to do most compiles. I only had to cc -T/usr/tmp to link Mg.

- The docs say that minix will use hard disk partitions larger than TOS's max of 16M. Using the atari HDX util, it will not let you create partitions > 15.9M (why atari does this BS is beyond me). Using the Supra utilities you can create partitions > 16M, but for some reason (that i have'nt looked into as yet) mkfs fails when you specify > 16K blocks (or it could very well be that i did something wrong). Everything works just fine for partitions < 16M (even with the Supra utils).
- I am having the hardest time trying to make a chmem'ed cem to run from Make. I guess the work around is to exec cc or maybe the passes themselves instead of doing a exec sh -c.

Coming up

Fix up Mg fully (special key support) - serial driver - I am working on Prabhaker to bring up Gulam. Send him fan mail!!

I guess that's it for now, after about a day and a half with minix. Congratulations are certainly in order to the authors. THANK YOU ast, johan and jost.

Bug Round-Up

Found two bugs in the Minix Atari ST distribution from PH:

- popen.c uses _exit(). _exit() is not in the library. Quick Fix: replace the one occurrence of _exit() with exit() (i know they are not the same, but it is close enough for government work, especially on the atari version where they have eliminated _cleanup)
- fdopen.c is missing. Here is one:

```
/*
 * They missed this in the standard ST distribution
 *   adapted from fopen.c
 *   ++jrb bammi@dsrgsun.ces.cwru.edu
 *
 */
#include <stdio.h>

FILE *fdopen(fd,mode)
int fd;
char *mode;
{
        register int i;
        FILE *fp;
        extern char *malloc();
        int flags = 0;

        /* find a table slot */
        for (i = 0; _io_table[i] != 0 ; i++)
                if ( i >= NFILES )
                return(NULL); /* if none avail */
        /* set up flags */
        switch(*mode){

        case 'w':
                flags |= WRITEMODE;
                break;
        case 'a':
```

```
                flags |= WRITEMODE;
                break;
        case 'r':
                flags |= READMODE;
                break;
        default:
                return(NULL);
        }

        if (( fp = (FILE *) malloc (sizeof( FILE))) == NULL )
                return(NULL);

        fp->_count = 0;
        fp->_fd = fd;
        fp->_flags = flags;
        fp->_buf = malloc( BUFSIZ );
        if ( fp->_buf == NULL )
                fp->_flags |= UNBUFF;
        else
                fp->_flags |= IOMYBUF;

        fp->_ptr = fp->_buf;
        _io_table[i] = fp;
        return(fp);
}
```

Structure, Part 4
By Colin Masterson

Choosing Good Data Types...

In our search for structured programs, choosing a good data type can be a great help.

The wonderful thing about languages like 'C' is the way they allow us to define, not just our own command verbs, but also our own types of data.

In BASIC we're restricted to strings, integers and reals. In 'C' we can define data types, structures, arrays of structures and pointers to arrays of structures to our hearts content. So let's do it !

Don't be afraid to create your own data types, in the end this too will help structuring.

How to create good data types is another story, but a word on when you should have them and how to use them is in place here.

Let's consider an example where we wish to draw an electronic circuit on the screen. One of the components we wish to draw will be a resistor. This will need a value, a name and the co-ordinates on the screen at which we want it to appear. (We may also want to include various other bits of information about the resistor too.)

We could create a set of arrays, or just a list of variables which we pass from function to function. However, since all these properties are very closely associated with this one item, it makes sense to create our own type which we will call a RESISTOR.

It would be reason able to have such a type which is a structure containing all the details.

In C this might be:

```
typedef struct {
        short line, col;    /*  screen position */
        long value;         /*  resistance in ohms  */
        char *name;         /*  name of resistor    */
        int flags;          /*  horizontal/vertical/power..*/
        } RESISTOR;
```

Now we can work with one variable instead of a list. We might, at higher level, be dealing with an array of resistors, for a Wheatstone bridge say, like this:

```
RESISTOR wheatstone_bridge[4];
```

This would be quite clear to someone reading it.

If we are coding a function to check if the bridge is in balance then, in 'C' at least, we can pass this array as a single parameter:

```
if(in_balance(wheatstone_bridge) == YES)
    :
    :
```

Pass a data structure and reduce the number of parameters and the meaning is kept quite clear. At the highest level we may wish to consider that our data type is a structure called CIRCUIT. This could well contain such information as supply voltage, logic family and a list of components. By creating the appropriate data type we have grouped variables which are closely related so we can handle them at a single step. We have raised the level of the programming language to one which suits our needs. In this case a language which understands resistors and which has commands like parallel(res1,res2). Variable names are chosen to reflect the meaning. As we move to a lower level we might have a need for a function which returns the nearest E24 value to a particular resistor.

```
give_E24(res)
RESISTOR res;
{
    :
    :
}
```

The low level functions deal with low level data types. Here we have not passed an array, but just a type RESISTOR. We could equally have just passed the value of the resistor, however this method keeps things consistent.

It is foolish to consider constructing a low level function to handle a high level data type. Foolish because changes to your data type will require severe changes to these low level functions and also because the syntax to extract a value from a structure within an array of structures to which you have received a pointer is easily miscoded.

Better to create loops like:

```
for(resistor = 0; resistor < 4; resistor++)
    E24[resistor] = give_E24(wheatstone_bridge[resistor]);
```

Any changes to the definition of the type RESISTOR need not result in a change to the function to give E24 values.

..and variable names!

Notice also that as the functions get lower in the hierarchy, the variable names can become simpler.

'res' indicating it's a resistor, any resistor, regardless of where it came from.

In 'C' common conventions are to use 'c' as a single character variable, 's' as a character pointer in low level functions. This is fine, but any occurrence of these single letter variables in other than the lowest level functions should be looked at critically.

Often, I will deliberately rename 'c' to 'token' in higher level functions just to emphasise the point.

Next time we'll look at what makes a good procedure.

Book Review: ADVANCED C

by Robert Schildt 19866, Osbourne McGraw-Hill, Berkeley, California, U.S.A. ISBN 0-07-881208-9. £19.95.

As the title suggests, this book is not suitable for the complete beginner. For the novice programmer, with a basic grasp of the language and the ability to operate a compiler, it provides C source code and background theory for a wide range of computing tasks.

For the more experienced, it shows that programming is more than technique. Through the detailed analysis of algorithms, the computer becomes an extension of the mind, a cerebral ROM cartridge, The book requires the support of a introductory text to the C language, such as Kernighan and Ritchie, and of course the documentation supporting the user's own C compiler.

After a very brief review of C language, Schildt leads us into the world of algorithms. Did you know that there are several ways of sorting character arrays, i.e. of arranging the characters into 'alphabetical' order? Schildt describes four methods in common use, and compares their effectiveness.

Then, the same algorithms are applied to sets of strings and more complicated entities. The aim, as always, is to close the eternity gap, the time which elapses between the pressing of a key and the appearance of output. Next, there are algorithms for searching.

Almost imperceptibly, we slide into data structures, the queues, linked lists and binary trees, various ways of organising information in the computer. By using these structures, multidimensional arrays can be prevented from consuming vast amounts of memory, leading to bigger and faster spreadsheets and the like. There is a chapter on expression parsing, programs that will accept expressions such as 10-5*3 and return the answer, in this case -5, and the parting of expressions containing variables. These routines form the backbone of all language compilers and interpreters, and spreadsheet programs. Schildt gives the source code for two different expression parsers.

In all, there are 337 pages for your money, abundant in source code. Oh yes, some other fragments are included; a lot on cyphers and a little on data compression, some statistics programs, a chapter on random number generation and simulations, how to access BIOS and DOS, how to convert BASIC and Pascal to C, and how to get portability of code. He used an Aztec C86 compiler for the IBM PC, And offers the programs in diskette form for $29.95 plus $5.00 shipping. I gave compiled several of the programs after keyboard entry on my own system (Sinclair QL, 640K RAM with RAM disk, Metacomco Lattice compiler) without difficulty.

Writing Portable 'C' programs. (Part I)
By Ron Wellsted.

What is "Portability"?

Probably the greatest strength of C is the ease with which programs can be moved from one computer to another with the minimum of alteration.

The majority of the issues affecting the portability of C programs come about from differences in either the compiler used or the target computer's run-time environment (operating system and/or hardware).

The Compile-Time Environment.

Differences between compilers are much smaller now than they were about five years ago. Nowadays, the only major variation is if the compiler is a "full K&R" implementation, or a subset, such as Ron Cain's Small C. Some differences are also found in which include files are present. However, stdio.h is almost always present (but not always!).

Beware of code written for some of the older compilers, as even the "full" compilers would exhibit major differences. As an example, consider the following code fragment:

```
int i;
struct t_struc *p;

i = p;
i->element = 0;
```

Obviously this code is totally invalid, right? Any attempt to compile it must result in several errors? Wrong, the "official K&R" UNIX Version 7 compiler would compile the above code without so much as a warning about incompatible data types. To compound the problem, if pointer occupies the same number of bytes as an int, the program would also run without any errors!

The most common area where compilers differ is with the order of evaluating expressions. For example try the following short program:

```
#include <stdio.h>

func(j)
int j, k;
{
 printf("j = %d, k = %d\n", j, k);
}

main()
{
 int i;

 i = 1;
 func(i++, i++);
}
```

You may find the result surprising! (Microsoft QuickC yields j = 2, k = 1).

Another major problem can occur with pre-processor macros. As most compilers will allow parameterised macros, some "functions" are actually macros. For example, Microsoft C implements toupper() et al. as macros. As a result

```
c = toupper(*p++);
```

will compile without error but will give unpredictable results when the program is run. (This is because the parameter, *p++, is used TWICE within the macro).

Fortunately, most of the problems of compiler portability have been or are being solved with the ANSI standard.

As an example, some Cpu's, such as the Motorola 68000 and the PDP-11, require that items larger than a byte be aligned on even address boundaries, others, such as the i8086 and the VAX-11 do not have this restriction. ANSI C allows the packing of structures to be controlled by the #pragma pack() directive. #pragma pack(1) will force byte alignment, #pragma pack(2) will force 2-byte alignment etc. #pragma pack() will return packing to the default.

In Part II, I will cover some of the problems associated with the run-time environment.

Review of Power C Trace by Martin Houston

Power C Trace is a comprehensive system for debugging C programs that have been compiled with the MIX Power C Compiler, (reviewed in the last issue of C Vu).

The program is available for IBM PC clones & close compatibles (at least to BIOS level) under MS or PC DOS.

Power C Trace comes with a 136 page manual. This review will only cover the simplest ways of using it. The product is packed with extra features I have not got the space to cover here. To do the program full justice would virtually involve re-writing the manual and that would leave very little room for anything else in C Vu.

This review concerns a run through of the following very simple C program:

```
/*
 * Program to test out MIX Power C Trace
 */
func1(s)
char *s;
{
     int i = 0;
     /* A simple 'strlen' function */
     while(*s++)
     i++;
     return(i);
}

main(argc, argv)
int argc;
char *argv[];
```

```
{
      int i;

      for(i = 0; i < argc; i++)
      {
      printf("arg %d is \"%s\" of length %d\n", i,
      argv[i], func1(argv[i]));
      }
}
```

To use C Trace the program to be debugged must first be compiled using the /t switch to the Power C compiler. For example:

```
pc /t /e test.c
```

This produces the following files:

TEST.MIX	.MIX object file (like .OBJ)
TEST.EXE	The executable program (will run on its own or with C Trace).
TEST.TRC	Special C Trace information for the source file TEST. There is a .TRC file for each source file.
TEST.SYM	Symbol table information for TEST.EXE

To run C Trace it is a simple matter of typing "pct test"

The test program will load and you are presented with a window on to the source code of the program with the first executable statement highlighted (the i = 0 of the for loop). To use C trace in its simplest way all you need to know is that pressing the space bar will cause your program to be executed one statement at a time. The hi-light bar will move to the statement that is about to be executed next each time the program is stepped. You will see the hi-lighting move from the for loop down to the printf statement and then to the body of func1 as it needs to be evaluated.

While in this mode you can use the up and down arrow keys to scroll the text of the program just as in an editor. Several commands are in fact provided for moving efficiently through the source window and searching for strings etc. In fact the only thing you cannot do is actually modify the program text that you see before you!

In addition to the space bar single step you have two more options on how the program is executed. If return is pressed the program runs at "trace speed" with the flow of control animated on the screen. There is also the option of running the program at full speed by pressing the X key but if you do this some of the more useful features of the debugger such as keeping of an execution profile will be lost.

Breakpoints in the code are inserted by moving a special 'breakpoint' cursor (a flashing red hi-light on a colour screen) with the cursor keys and pressing the INS key to make a breakpoint at a statement. Reaching a breakpoint will cause execution to stop, even if the program has been running at full speed. Breakpoints can be deleted with the DEL key when no longer required.

The source code is only one of seven 'windows' offered by C Trace. These windows can be displayed on the screen up to four at a time in any desired combination and positioning or singly taking up the whole screen. The windows are: source, program output, variables, watch points,

memory, symbols and assembly. Of these the first four are the most commonly used.

The source window has already been introduced. Its function is to show the program in source form and the current location of the program counter.

The output window is self-explanatory; if you program wants to print anything to the screen it has to go somewhere. The output window is also where you type data at the program if it is interactive. The output window can be in graphics mode so graphics programs can be debugged with C Trace (but must be full screen only in this case).

The variables window and watchpoint window are best considered as a pair. The variables window displays all the program variables with their current values The list of variables can be scrolled through in the same way as the source window. When the program is run at trace speed the values of the variables change before your very eyes as the program modifies them. Watchpoints are to variables are as breakpoints are to the program code. They are selected in the same way as breakpoints, by positioning a cursor in the variables window and pressing the INS key. When a variable is selected to be a watch point a menu pops up asking on what condition execution is to be stopped when that variable is modified. You can select between ANY change and the value becoming less than, greater than or equal to a specified value. An example from the above program would be a watchpoint set on i becoming 4 so the program could be quickly run through the first four arguments and then single stepped through the next. This feature is a boon if you find some data is being corrupted and want to find which part of the program is the culprit!

The memory window gives you the machine's view of what is going on. The raw memory space of the program is displayed in a variety of formats so that you can check exactly how data is set out etc. Memory locations can also have watch points set on them so data can be watched even if it has no symbolic name. Changes can be made to anything in the memory for running repair bug fixes and tests.

The symbols window lists all the external symbols that are known about from the object file. Breakpoints can be placed on any of these symbols, once again using the INS key, so that references to library routines etc. can be trapped on.

The assembly window is very similar to the source window except it shows Intel assembly language instead of C source code. If you don't know assembler then this will not be a great deal of help but for those that can understand it the window is very useful in that:

1) Breakpoints can be inserted at a single machine instruction level.
2) ANY code can be so traced - not just C source files that have been compiled with the /t flag
3) If the C and Assembler windows are displayed on the screen together then the execution cursors of both move together as the program runs. The relationship between the C source and what the compiler produces can be seen very clearly. Coupled with a book to explain the assembler mnemonics it is an excellent way to teach yourself assembler programming! The code generated by our simple example above should be quite easy to follow in assembler with the aid of constant reference back to the C source with the context always indicated.

In this article I have only just scratched the surface of what C Trace can do. The best way to find out more is to go out and buy it! That brings us on to the subject of prices - usually at this point in a review you think that the product was quite interesting but a price tag of hundreds of pounds makes that interest wain very quickly. You have no need to worry on that score with C Trace. MIX Software has a policy of pricing software so that the HOME user can afford it. Power C Trace costs

just £19.95 (inclusive of VAT and P&P). For that you get the program, lots of examples to work through and a 136 page paperback manual. The Power C compiler (reviewed in the last issue of C Vu) is still just £24.95 inclusive. ALL library source AND the Power C Assembler is available for just another £10.00. That makes a development tool set for MS-DOS C programming for under £55.

No royalties are payable for programs developed under Power C so it is a great product for the amateur programmer producing next years 'blockbuster' on a tight budget.

Further details from:

Analytical Engines Ltd
P.O. BOX 35 Eastleigh,
Hampshire,
SO5 5WU
Tel (0703) 262099

Colossal Caves Hists The QL
by Maurice Watson

Maurice Watson has done all the hard work for you in getting 'Colossal Caves' ported to the QL

The problems he encountered during this make interesting reading so I reproduce his documentation here. The QL adapted version of Vol 21 is available as the new library volume 26. As Maurice says below; if you would like to get this software on QL disk format directly then write to him at 69 Windsor Road, Gravesend, Kent DA12 5BW.

> Dear Martin, 2.7.88

Beneath this file are numerous others which represent a translation of volume 21 of the library, games#2, the Colossal Caves adventure game, for the QL. You may include it in your QL library if you wish. If you are not forming a separate library, I am prepared to supply copies of this disc under the normal terms and conditions.

It has been tremendous fun, and I like the game. It was a fight however to squeeze it through Metacomco Lattice 3.01. If you look at my version of the doc file, or at some of the source code, you will see why. I am not sure if it is practicable to go much further with the library until these problems with the compiler are resolved. I wonder if you can offer any suggestions for a way forward?

Yours sincerely,
Maurice Watson
28 JUNE 1988

Notes on ADVENTURE (Sinclair QL version)

1) TO PLAY ADVENTURE

The ADVENTURE game requires one single sided drive & 96K of memory. ADVENT1_O_BIN, ADVENT1_TXT, ADVENT2_TXT, ADVENT3_TXT, ADVENT4_TXT should be present on

flp1_.

The game may be started by typing CRUN_W"flp1_ADVENT1_O_BIN<cr>". A saved game may be restarted by typing CRUN_W"flp1_ADVENT1_O_BIN -r<cr>". Debug data will be output by typing CRUN_W"flp1_ADVENT1_O_BIN -d -d<cr>".

Alternatively, and more conveniently, use the Superbasic program flp1_BOOT to start the program. Load this program and then type RUN<cr>, followed by y<cr> for a new game or n<cr> for a restart.

2) TO HACK ADVENTURE

The ADVENTURE game source files are either, header, code or text files.

ADVENT_C	- in four sections -initialization, save game, restore game
ENGLISH_C	- interpret game player's commands
DATABASE_C	- in three sections - text file management & output
ITVERB_C	- intransitive verbs execution
VERB_C	- in five sections - transitive verbs execution
TURN_C	- in five sections - analysis & execution of player's commands
ADVENT0_C	- utility to create "ADVTEXT_H" file
ADVENT1_TXT	- long cave description
ADVENT2_TXT	- short cave description
ADVENT3_TXT	- long & short object description
ADVENT4_TXT	- conversational descriptions & responses
ADVENT_DOC	- this ADVENTURE documentation file
QDOS_H	- extended header file
ADVENT_H	- #define & structure statements
ADVCAVE_H	- cave & travel arrays
ADVTEXT_H	- TXT file message indexes
ADVDEF_H	- data constants & variables definitions
ADVDEC_H	- data constants & variables declarations

Sinclair version only:-

These files have been used to compile certain files, which would otherwise have generated "out of memory" errors, due to very low source code capacity of the compiler (not the system!). This is a most serious limitation of Metacomco Lattice (version 3.01).

ADVDATA1_C	- dummy module 1 used to compile location array
ADVDATA2_C	- dummy module 2 used to compile first part of vocabulary
ADVDATA3_c	- dummy module 3 used to compile cave array
ADVDATA4_C	- dummy module 4 used to compile second part of vocabulary

ADVWD_H	- first part of vocabulary
ADVWD2_H	- second part of voabulary
ADVNT_H	- contracted version of advent_h
ADVT_H	- ditto

Finally,

PROG_LINK	- control file for linker
ADVENT1_O_BIN	- compiled game program
ADVENT0_O_BIN	- compiled text file modification program

WARNING:

The TXT files are the ASCII text messages used throughout the game. After any changes, they must be reconstructed using the utility program, ADVENT0_O_BIN (source ADVENT0_C). This program creates the header file ADVTEXT_H which is "#include"d into "ADVDATA1_C" during compilation. After ANY changes to the "TXT" files, recreate a new "ADVTEXT_H" file. Make sure that the four "TXT" files and the utility "ADVENT0_O_BIN" are on the default drive.

HISTORICAL - THE IBM PC VERSION

The game was translated from BDS C to CII C86 and standardized as per UNIX standard i/o library functions. The following changes were instituted:

0) UNIX standard i/o
1) "include"d header files & "extern"al statements added
2) cave/travel data arrays are now internal
3) word/code data arrays are now internal
4) TXT message index arrays are now internal
5) TXT file format doesn't require # terminator character
6) save & restore game overlays intergrated with "ADVENT.C"
7) word/code syntax parsing optimized in "ENGLISH.C" BINARY LEX-ORDERED WORD LOOK-UP added in "DATABASE.C"
8) TXT message indexing & output optimized in "DATABASE.C"
9) TXT message typos corrected
10) created "ADVENT0.C" utility (cf. #1, #4 & #5)
11) created "ADVENT.DOC" documentation file

The IBM modifications described above were implemented by: Jerry D. Pohl 1922 Junction

Avenue San Jose, CA 95131 (408) 298-1262 / (408) 298-3185 (both 8..6, m..f)

CHANGES MADE IN SINCLAIR QL VERSION

Play variables, originally declared as external integers, have been collected into a play structure, a pointer to which is passed as a parameter through all the modules. English variables and file pointers have been packaged similarly. This makes flow of information more explicit.

Some of the files must be subdivided to enable them to be compiled using the Metacomco QLC Development Kit, version 3.01 (1985). Four new modules, advdata1-4 inclusive, have been used to bring in pointers to the main data arrays. Cut-down versions of header files have been used when necessary . Details given above .The resulting object files are linked using the GST linker provided in the kit.

Some attention has been given to the presentation of the output, by defining screen areas. The game save and restore facilities have been rewritten to eliminate read and write errors.

An extended qdos_h header file containing several useful addresses, as well as those used in the game, has been included. The facilities of the QDOS operating system have been used to supplement standard kit routines where necessary.

Bytes removed from all source and header files, as they are not required on this system. Text files reconstructed accordingly.

These changes were made by:

Maurice Watson 69 Windsor Road, Gravesend Kent, DA12 5BW. 0474-568335. June 1988.

Do You Use DR-CDOS?

If so Carl Hibbard (Membership #8713) would like to hear from you

“I am particularly interested to find out if there is anyone doing any large developments using the CDOS operating system.

I am using Digital Research's compiler which isn't brilliant, and wonder what else anybody recommends.

I have access to a variety of machines, so I can use a PC to develop bits, but I would like to stay with the confines of the lovely CDOS system.

Most of my work is done using 8086 assembler, and have only just started using C, so your magazine is a great help.

If you know of anyone, could you please let me know, thanks very much.

How To Get In Touch

If you are also using CDOS & would like to get in touch with Carl then drop Martin Houston a line (the address is, as always, on the front page).

If you have a particular interest, and would like the rest of the group to know about it (!) then drop us a line as well.

Small-C/Plus for the Z80 by Ron Yorston.

Ron Yorston has been busy recently, improving the Small-C compiler. His efforts can by found on library volumes 23a and 23b

Small-C/Plus is based on 'Small-C compiler with floating point' from SIG/M Volume 224. The main changes that Ron has introduced are the following:

- Additional Control Structures
- More pre-processor directives
- Structures and Unions
- More data types
- Improved Code generation
- Enhanced library (much re-written in assembler)
- Library management and library reference resolution
- Assembly code optimiser

Ron writes: "I did my program development on an Amstrad PCW 8256, but the enclosed disks are MS-DOS format. There are four directories on the disks: SMALLCPL.1A, SMALLCPL.1B, SMALLCPL.2A and SMALLCPL.2B. Each of these directories will fit onto one side of an Amstrad CF2 disk. I've included the full source code for the compiler, library and the utilities I write myself. In addition there are compiled versions suitable for use on the Amstrad. I've successfuly got the compiler to work as a cross-compiler on a UNIX system and on a PC."

23a\smallcpl.1a:

This disk contains the source for the Small-C/Plus library, header files, and a complete copy of the library.

23a\smallcpl.1b:

This disk contains miscellaneous utilities for the Small-C/Plus compiler

23b\smallcpl.2a:

This disk contains the source for the Small-C/Plus compiler:

23b\smallcpl.2b:

This disk contains the Small-C/Plus compiler and documentation

Book Review: Simple 'C' - A Beginners Guide Reviewed by Martin Houston

Ian Sinclair, David Fulton Publishers ISBN 1-85346-057-5 180pp #12.95

This is a book aimed at microcomputer users who 'cut their teeth' on BASIC and now wish to move on to C.

It starts by explaining how the C language fits in with other computer languages: "C allows you the control over what the computer does that you normally associate with machine code, and will generate compiled code that is almost as compact as machine-code from an assembler. At the same time, though, C provides all the structures of a good high level language, like the facilities for creating loops, many different variable types, structured variables like arrays, and so on".

Assuming that the reader has some prior knowledge of one computer language (even if its only BASIC!) allows this book to get going much faster than would be possible in a raw beginners text. Where C differs from BASIC, the author explains the differences and why the way C is organised is better. BASIC programmer are weaned G off what Mr Sinclair calls the 'drunken fly' approach to program writing. The splitting of a C program into functions naturally encourages a structured design far better than ordinary unstructured BASIC can.

The book starts with simple input and output programs using the standard library functions that should be the same on all compiler systems. All the example programs in the book are small, a page at the most, so should not take too long to work through.

Thorny areas such as pointers and structures are covered in some detail and some standard library functions are described. The book does not claim to be a complete reference work, but it would serve as a good introduction for a BASIC programmer wishing to move on to C.

The examples in the book assume the Zortech C compiler on an Amstrad PC. As the author points out, this is a very common configuration with a standard way of going about things so adapting the examples to other systems should not be too much of a problem.

THE MAN
by Colin Masterson

And during the seven good years when grass was a vivid green and contracts were fat and the workers were busy and applications large and data types complex, there came a summons unto the Man (for so was the project manager known) to visit with those above. And with a clear heart and an open mind did he visit with them and they enlightened the Man as to his great mission in life.

And so the Man did leave the presence of the Great ones above with a heavy heart and considerable thinking.

And that next day, did the Man summon unto his feet all his disciples and followers and they did gather about him in awe and expectation.

"I have spoken with the Great ones and our mission is clear", spake the Man.

And the followers and disciples listened in silence. "And the need is for a great application which shall be heavy with data input."

And a gasp went out from those close to the Man.

"And the types of data shall be many and of such variety as we may only try to conceive", sayeth the Man. And yet the crowd remained still.

And the Man considered his task and the way became clear to him, and he said. "The way is clear and the need is for a good text input function." And some two brace of his closest disciples departed to commence their task.

Further did the Man speak saying: "And the way is clear and the need is for a good numeric input function with ability to handle the many and varied types and forms of data we shall receive." And a further brace of disciples departed knowing their task.

And quietly the man considered and the days did pass. Then did the disciples return bearing text and numeric input functions and knelt before the Man laying them before him.

And they were pleased.

And the Man looked closely at their gifts and sayeth:

"Yet the amount of data is of such size that it will not all have capability to be held upon one single screen." And he directed his people to prepare a window library.

And, after some little days, did his disciples return presenting him, most proudly, with a window library.

And the Man looked at the text input function.

And the Man looked at the numeric input function. And the Man looked at the window library. And he was much troubled.

And so the Man considered again the great task and was suddenly beckoned before the Great ones (For so the monthly project meetings come along) and he laid himself bare before them. And they could see that the fruits of his labours were good and worthy and they could see that the task was all the while nearer.

But the Great ones reminded the Man of the guiding light, and that light shall be paramount. And that guiding light shall be ease of data input. And they sent the Man back to his flock.

And the Man again was troubled and considered the great guiding light and said to his flock.

"So do we need a window management function that we might contain the data input functions and control the window library and handle the vast and varied data for which forms our great task."

One score days and one score nights did all his flock and followers and disciples work. Until, at last, they had a window management function and they were pleased, an so took it to the Man.

And the Man looked at his text input function, and the Man looked at his numeric input function, and the Man looked at the window library and at the management function that the people had made. And he saw that it was good.

And the followers sat quietly that they might better hear the Man's praise. And the Man was silent. And the people fell into a great gloom. For it was clear to the man that it was not enough.

For the great guiding light showed clearly that each window must have its own data handling function.

And the guiding principle of ease of data entry was close, but not yet his.

And he wondered.

And the disconnection between the functions, and the fundamental needs of generality and reusability, which were in his bones, were both clear to him.

And he wondered.

And, as he wondered, his knowledge and understanding of the great guiding light became complete. For clearly could he see that each function must capture all commands and signals and cases for all their brothers. And the conflict between this and generality distressed the Man, and he grew thin. And the need to handle back stepping and helpful actions was also obvious to the Man, and he was much troubled.

And there came at that time a crashing in the sky and a renting in the clouds and so did the clouds tear apart. And falling from the great rent in the clouds did appear a token . Bright and shining in the sun. And at once the way was clear to the Man. And he paid thanks to the sky above and went in great haste to assemble his flock.

And it was clear to those assembled that a great change had overcome the Man. And they were at once frightened and expectant. And the Man spoke.

"The way is simple, and the guiding principles of ease of data entry and window libraries and input functions can be united."

And the people were in awe and they cried out:

"How ? Tell us Man, how this can be ?"

And the Man spoke again.

"Each function shall obey certain commandments", sayeth the Man, "and these commandments shall control the passing of tokens . And a token shall be the data. And the functions shall ask, 'is this my token ?' and if the answer shall be yes, then the function will act upon it.

And if the answer shall be no, then the function shall pass on the token to the next function."

And the Man continued:

"So shall each function be contained within, and connected to, the last. And functions shall inform their caller if they be happy with the token, or if they be sad and have no use thereof. And each function shall recognise a happy or an unhappy call, and so shall they know how the token has been used, and, indeed, if the token might be passed on again to another function."

And the place was still as the people considered.

And the Man was silent and waited.

And, one by one, so could the Man see that the people understood.

And again the Man spoke saying:

"And furthermore, shall each function be required to check if it hath received a token belonging to a higher heirarchy, and, if this be the case, so shall he yield this token . And each function need only concern with a small number of tokens - for which they have a rightful and useful need - passing all others onwards."

And the Man could see that the guiding principle of ease of data entry had been upheld. And he could feel in his bones the goodness of the generality of the functions, and soon could he see the structure and hierarchy of what he, and his people had created.

And the Man was at last pleased.

And he sat with his people and his people were pleased.

And at last was the man summoned again to the Great ones and they sook of him his progress.

And the Man spoke loud and long to the Great ones and they listened.

And they were pleased.

And at last a Great one spoke saying: "Bloody good job Simpkins".

Writing For C Vu
by Phil Stubbington

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!

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:* _____

THE SPECIAL OFFER.... FREE LIBRARY VOLUMES!

Every issue of C Vu so far has carried a special offer of some sort – and this is no exception. We are offering you the chance to receive volumes from the Source Library (detailed elsewhere in this issue) for the price of a stamp.

All you have to do is submit an article, review or letter for publication in C Vu. The only rules that apply, in addition to the guidelines set out in "Writing For C Vu" are:

- each submission must be on a seperate disk - 5 1/4" or 3 ½"
- the _minimum_ length is 500 words

If you article appears in a subsequent issue of C Vu, you will receive the library volume(s) of your choice. Each accepted submission entitles you to one volume - so if you write a five part series, you can receive five volumes.

You could, in fact, get all the volumes in the library for next to nothing by writing a review of each volume (perhaps with details of any difficulties you encountered if you are porting to a different compiler, O.S., etc.)

Quick Tip #1 By Phil Stubbington

Debugging GEM Desk Accessories

Ever wondered how on earth you are supposed to debug desk accessories? Well, the simple answer is don't! With a few exceptions, it is much easier to debug them as applications, using the pre-processor to include or exclude the DA specific bits of code. For example, where you have the menu_register call, do the following:-

```
#ifdef ACC
 MenuBarID = menu_register(gl_apid," AutoSave");
#endif
```

When running as a "normal" program, you won't of course ever receive the AC_OPEN message, so wherever you catch events something like this is needed to simulate it:-

```
void CatchEvents()
{
int EventReceived;

#ifdef ACC
  EventReceived = evnt_multi(MU_MESAG|MU_TIMER,
  0,0,0,0,0,0,0,0,0,0,0,0,0,
  MessageBuffer,
  TIMERLO,TIMERHI,
  &dummy,&dummy,&dummy,&dummy,&dummy,&dummy);
  if(EventReceived & MU_MESAG)
DoMessages();
  if(EventReceived & MU_TIMER)
DoTimer();
#else
  DoMessages();
#endif
}
```

And the DoMessages function will look something like this:-

```
void DoMessages()
{
#ifdef ACC
 if(MESSAGE_TYPE == AC_OPEN)  {
#endif
  /* whatever code is normally executed in response to an AC_OPEN */
#ifdef ACC
  }
#endif
}
```

The greatest difference between DA's and standalone programs is of course that DA's never terminate, so although we have an "appl_init" call, we don't need (and in fact should never reach) the corresponding "appl_exit" call. So some changes need to be made to your "main" function:-

```
void main()
{
  appl_init();
  InitialEyes();
#ifdef ACC
  while(TRUE)
#endif
CatchEvents();
#ifndef ACC
  appl_exit();
#endif
}
```

Then, if you want to debug your GEM DA's, you'll have to do something like:-

```
 cc -UACC ....
```

or include the following line:-

```
 #undef ACC
```

in the appropriate source files. Finally, do remember to follow your documentation when it comes to creating DA's - the startup code (i.e. the bit that is executed before "main") is nearly always different.

WHY DON'T YOU...?

Send in your own quick tip? Quick tips can be on any subject that might be of interest to members, and should be up to 1000 words in length. See the article on "Writing for C Vu" earlier in this issue for guidelines.

Professional GEM
by Tim Oren

Part One, and the topic is windows...

HELLO, WORLD!

For those whom I have not met in person or electronically, an introduction is in order.

I am a former member of the GEM programming team at Digital Research, Inc., where I designed and implemented the GEM Resource Construction Set and other parts of the GEM Programmer's Toolkit. I have since left DRI to become the user interface designer for Activenture, a startup company which is developing CD-ROM technology for use with the Atari ST and other systems.

The purpose of Professional GEM is to pass along some of the information and tricks I have accumulated about GEM, and explore some of the user interface techniques which a powerful graphics processor such as the ST makes possible.

GROUND RULES

I am going to assume that you have both a working knowledge of the C programming language and a copy of the ST Programmer's Toolkit with documentation (available from Atari). If you lack either, don't panic. You can read the columns to get the flavor of programming the ST, and come back for a more serious visit later on.

For now, I will be using code samples that will run with the Atari-supplied C compiler, also known as DR C-68K, or Alcyon C. I will be using the portability macros supplied with the Toolkit, so that the code will also be transferable to other GEM systems.

Both of these items are subject to change, depending on reader feedback and the availability of better products.

If you do not have a copy of the source to the DOODLE.C GEM example program, you should consider downloading a copy from SIG*ATARI. Although it is poorly documented, it shows real-life examples of many of the techniques I will discuss.

Getting started with a windowed graphics system seems to be like getting into an ice-cold swimming pool: it's best done all at once.

Anyone who has looked at "Inside Macintosh" has probably noticed that you have to have read most of it to understand any of it. GEM isn't really much different. You have all the reference guides in your hand, but nothing to show how it all works together.

I am hoping to help this situation by leading a series of short tours through the GEM jungle. Each time we'll go out with a particular goal in mind and follow the path that leads there. We'll look at the pitfalls and strange bugs that lurk for the unwary, and show off a few tricks to amaze the natives. The first trip leaves immediately; our mission is to get a window onto the ST screen, with all of its parts properly initialized.

WE DO WINDOWS

One of the most important services which a graphics interface system provides for the user and programmer is window management.

Windows allow the user to perform more than one activity on the same screen, to freely reallocate areas of the screen for each task, and even to pile the information up like pages of paper to make more room. The price for this increased freedom is (as usual) paid by you, the programmer, who must master a more complex method of interacting with the "outside world".

The windowing routines provided by ST GEM are the most comprehensive yet available in a low-cost microcomputer. This article is a guide to using these services in an effective manner.

IN THE BEGINNING

In GEM, creating a window and displaying it are two different functions. The creation function is called wind_create, and its calling sequence is:

```
handle = wind_create(parts, xfull, yfull, wfull, hfull);
```

This function asks GEM to reserve space in its memory for a new window description, and to return a code or "handle" which you can use to refer to the window in the future. Valid window handles are positive integers; they are not memory pointers.

GEM can run out of window handles. If it does so, the value returned is negative. Your code should always check for this situation and ask the program's user to close some windows and retry if possible. Handle zero is special. It refers to the "desktop", which is predefined as light green (or gray) on the ST. Window zero is always present and may be used, but never deleted, by the programmer.

The xfull, yfull, wfull, and hfull parameters are integers which determine the maximum size of the window. Xfull and yfull define the upper left corner of the window, and wfull and hfull specify its width and height. (Note that all of the window coordinates which we use are in pixel units.)

GEM saves these values so that the program can get them later when processing FULL requests. Usually the best maximum size for a window is the entire desktop area, excepting the menu bar. You can find this by asking wind_get for the working area of the desktop (handle zero, remember):

```
wind_get(0, WF_WXYWH, &xfull, &yfull, &wfull, &hfull);
```

Note that WF_WXYWH, and all of the other mnemonics used in this article, are defined in the

GEMDEFS.H file in the ST Toolkit.

The parts parameter of wind_create defines what features will be included in the window when it is drawn. It is a word of single bit flags which indicate the presence/absence of each feature. To request multiple features, the flags are "or-ed" together. The flags' mnemonics and meanings are:

NAME	A one character high title bar at the top of the window.
INFO	A second character line below the NAME.
MOVER	This lets the user move the window around by "dragging" in the NAME area. NAME also needs to be defined.
CLOSER	A square box at the upper left. Clicking this control point asks that the window be removed from the screen.
FULLER	A diamond at upper right. Clicking this control point requests that the window grow to its maximum size, or shrink back down if it is already big.
SIZER	An arrow at bottom right. Dragging the SIZER lets the user choose a new size for the window.
VSLIDE	defines a right hand scroll box and bar for the window. By dragging the scroll bar, the user requests that the window's "viewport" into the information be moved. Clicking on the gray box above the bar requests that the window be moved up one "page". Clicking below the bar requests a down page movement. You have to define what constitutes a page or line in the context of your application.
UPARROW	An arrow above the right scroll bar. Clicking here requests that the window be moved up one "line". Sliders and arrows almost always appear together.
DNARROW	An arrow below the right scroll bar. Requests that window be moved down a line.
HSLIDE	These features are the horizontal equivalent of the RTARROW above. They appear at the bottom of the window. Arrows LFARROW usually indicate "character" sized movement left and right. "Page" sized movement has to be defined by each application.

It is important to understand the correspondence between window features and event messages which are sent to the application by the GEM window manager. If a feature is not included in a window's creation, the user cannot perform the corresponding action, and your application will never receive the matching message type. For example, a window without a MOVER may not be dragged by the user, and your app will never get a WM_MOVED message for that window.

Another important principle is that the application itself is responsible for implementing the user's window action request when a message is received. This gives the application a chance to accept, modify, or reject the user's request.

As an example, if a WM_MOVED message is received, it indicates that the user has dragged the window. You might want to byte or word align the requested position before proceeding to move the window. The wind_set calls used to perform the actual movements will be described in the next

article.

OPEN, SESAME!

The wind_open call is used to actually make the window appear on the screen. It animates a "zoom box" on the screen and then draws in the window's frame. The calling sequence is:

```
wind_open(handle, x, y, w, h);
```

The handle is the one returned by wind_create. Parameters x, y, w, and h define the initial location and size of the window. Note that these measurements INCLUDE all of the window frame parts which you have requested. To find out the size of the area inside the frame, you can use

```
wind_get(handle, WF_WXYWH, &inner_x, &inner_y, &inner_w, &inner_h);
```

Whatever size you choose for the window display, it cannot be any larger than the full size declared in wind_create.

Here is a good place to take note of a useful utility for calculating window sizes. If you know the "parts list" for a window, and its inner or outer size, you can find the other size with the wind_calc call:

```
wind_calc(parts, kind, input_x, input_y, input_w, input_h,
          &output_x, &output_y, &output_w, &output_h);
```

Kind is set to zero if the input coordinates are the inner area, and you are calculating the outer size. Kind is one if the inputs are the outer size and you want the equivalent inner size. Parts are just the same as in wind_create.

There is one common bug in using wind_open. If the NAME feature is specified, then the window title must be initialized BEFORE opening the window:

```
wind_set(handle, WF_NAME, ADDR(title), 0, 0);
```

If you don't do this, you may get gibberish in the NAME area or the system may crash. Likewise, if you have specified the INFO feature, you must make a wind_set call for WF_INFO before opening the window.

Note that ADDR() specifies the 32-bit address of title. This expression is portable to other (Intel-based) GEM systems. If you don't care about portability, then &title[0], or just title alone will work fine on the ST.

CLEANING UP

When you are done with a window, it should be closed and deleted. The call

```
wind_close(handle);
```

takes the window off the screen, redraws the desktop underneath it, and animates a "zoom down" box. It doesn't delete the window's definition, so you can reopen it later.

Deleting the window removes its definition from the system, and makes that handle available for reuse. Always close windows before deleting, or you may leave a "dead" picture on the screen. Also be sure to delete all of your windows before ending the program, or your app may "eat" window handles. The syntax for deleting a window is:

```
wind_delete(handle);
```

THOSE FAT SLIDERS

One of ST GEM's unique features is the proportional slider bar. Unlike other windowing systems, this type of bar gives visual feedback on the fraction of a document which is being viewed, as well as the position within the document. The catch, of course, is that you have two variables to maintain for each scroll bar: size and position.

Both bar size and position range from 1 to 1000. A bar size of 1000 fills the slide box, and a value of one gets the minimum bar size. To compute the proper size, you can use the formula:

```
size = min(1000, 1000 * seen_doc / total_doc)
```

Seen_doc and total_doc are the visible and total size of the document respectively, in whatever units are appropriate. As an example, if your window could show 20 lines of a 100 line text file, you should set a slider size of 200. Since the window might be bigger than the total document at some points, you need the maximum function. If the document size is zero, force the slider size to 1000. (Note: You will probably need to do the computation above with 32-bit arithmetic to avoid overflow problems.)

Once you have computed the size, use the wind_set function to configure the scroll bar:

```
wind_set(handle, WF_VSLSIZE, size, 0, 0, 0);
```

This call sets the vertical (right hand) scroll bar. Use WF_HSLSIZE for the horizontal scroller. All of these examples are done for the vertical dimension, but the principles are identical in the other direction.

Bar positioning is a little tougher. The most confusing aspect is that the 1-1000 range does not set an absolute position of the bar within the scroll box. Instead, it positions the TOP of the bar within its possible range of variation.

Let's look at our text file example again to make this clearer. If there are always 20 lines of a 100 line file visible, then the top of the window must be always be somewhere between line 1 and line 81. This 80 line range is the actual freedom of movement of the window. So, if the window were actually positioned with its top at line 61, it would be at the three-quarter position within the range, and we should set a scroll bar position of 750. The actual formula for computing the position is:

```
pos = 1000 * (top_wind - top_doc) / (total_doc – seen_doc)
```

Top_wind and top_doc are the top line in the current window and the whole document, respectively. Obviously, if seen_doc is greater or equal to total_doc, you need to force a zero value for pos. This calculation may seem rather convoluted the first time through, but is easy once you have done it. When you have computed the position, wind_set configures the scroll bar:

```
wind_set(handle, WF_VSLIDE, pos, 0, 0, 0);
```

WF_HSLIDE is the equivalent for horizontal scrolling.

It is a good practice to avoid setting the slider size or position if they are already at the value which you need. This avoids an annoying redraw flash on the screen when it is not necessary. You can check on the current value of a slider parameter with wind_get:

```
wind_get(handle, WF_VSLIDE, &curr_value, &foo, &foo, &foo);
```

Foo is a dummy variable which needs to be there, but is not used. Substitute WF_VSLIDE with whatever parameter you are checking.

One philosophical note on the use of sliders: It is probably best to avoid the use of both sliders at once unless it is clearly appropriate to the type of data which is being viewed.

Since Write and Paint programs make use of the sheet-of-paper metaphor, moving the window around in both dimensions is reasonable. However, if the data is more randomly organized, such as a tableau of icons, then it is probably better to only scroll in the vertical dimension and "reshuffle" if the window's width is changed. Then the user only needs to manipulate one control to find information which is off-screen. Anyone who has had trouble finding a file or folder within a Desktop window will recognize this problem.

COMING UP NEXT

In my next column in Antic Online, we'll conclude the tour of the ST's windowing system. I'll discuss the correct way to redraw a window's contents, and how to handle the various messages which an application receives from the window manager. Finally, we'll look at a way to redesign the desktop background to your own specifications.

Which C 4 Me?
by Phil Stubbington

A quick guide to C compilers currently available for the Atari ST

Of the 68000 machines currently available, the Atari ST has probably the widest variety of C compilers available for it. This can be both an advantage (greater choice) and a disadvantage (which one for me?). In this potted guide, I hope to cover the major features of the compilers currently available, and hopefully steer you in the right direction!

For the home user (i.e. someone who doesn't have to make a living out of it!) your best bet, if you just want to find out whether C suits you or not, then the public domain is the obvious place to go. Most PD libraries have a subset C compiler available, but don't expect any bells and whistles. Being a subset compiler, not all the features of C are supported (floating point and structures are usually missing), and you are unlikely to find a PD compiler which supports GEM. Running on floppies only is going to cause a few headaches (although this is also true of most compilers - public domain or not).

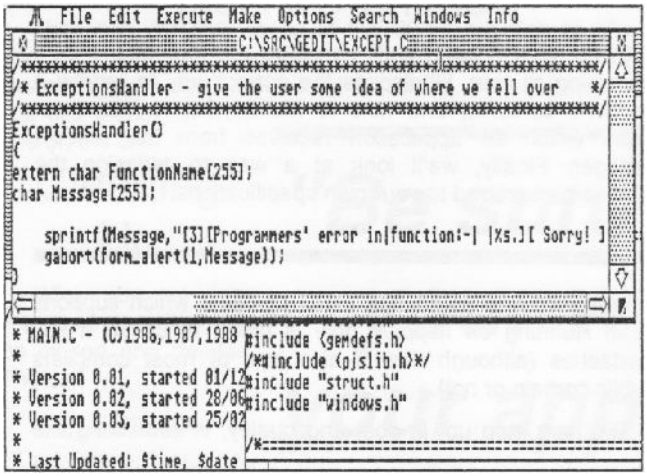
The next step up, in cost and quality, is something like the GST C compiler. This is, as far as I can tell, a tarted up version of Small-C (a public domain compiler by L.E. Payne and J.E. Hendrix) - i.e. no floating point, and no structures. It does support most GEM functions, and is GEM-based. At around œ15 (you may find it even cheaper!) it is probably the best introduction to C you are likely to get for the money.

The first full C compiler is Lattice C from MetaComCo. It is also the cheapest full development system (i.e. you get the compiler, text editor, resource file editor, linker, debugger, shell and documentation in one package). The lowest price I've seen is around £70. It may be worth hanging on for the new version before taking the plunge (upgrades cost around £35) as you'll get a much better text editor (Tempus) and a faster linker.

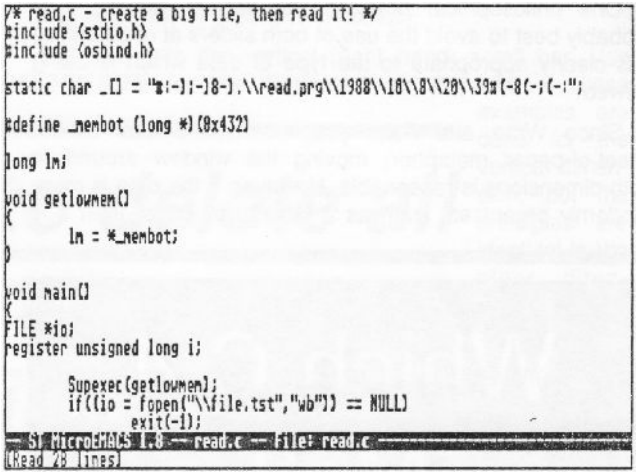
The next alternative is Prospero C at around £100. If I was in the market for a C compiler at the

moment, this is the one I would go for. Although you will have to buy a resource editor separately (another £25 - £30) it is a much better product than Lattice, and being a British based company, support is much better. It is also, as far as I know, the only C compiler for the ST that comes up to the ANSI standard. This adds features like function prototyping which can cut down debugging time significantly.

At around the same price, Mark Williams C does include a resource editor (a version of K-Resource), so no extra expenditure is necessary. MWC is not GEM-based, and contains a version of the MicroEmacs text editor, which I don't personally like (too cryptic by half).

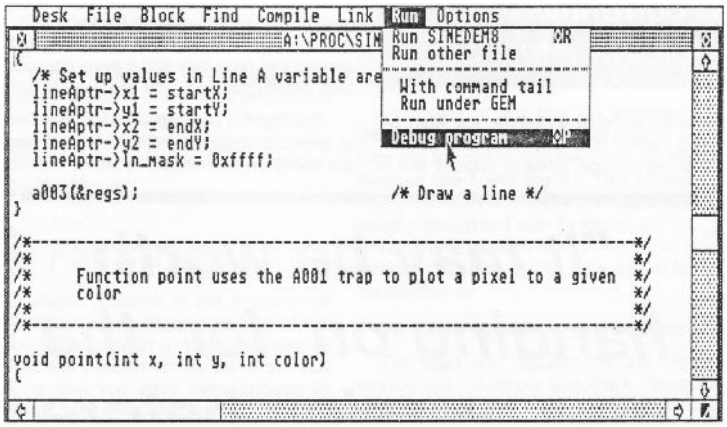


Megamax Laser C in action - probably the fatest ST compiler in the world. Also the text editor 99% of C Vu editors prefer - i.e. extremely fast, but low on features.



A version of MicroEmacs comes wit Mark Williams C - not my favourite editor, but popular with the "real programmers read ex dump" brigade.

The only C development system which can happily run from floppies is Megamax Laser C, but unless you are willing to buy direct from the States (where it can cost as little as $125 – around £70) you may be expected to pay as much as £160. I reviewed Megamax Laser C in the last (as in final) issue of the Moving Finger, which you may be able to get hold of. The resource editor is only around 90% complete, but other than that, it is well worth £70 - £80. It is also the fastest system of the lot.



Prospero C is the newest ST C compiler to hit the market. Probably my #1 choice, even though it lacks a resource editor. Scrolling could be faster as well.

The next step up is Manx Aztec C. Two versions are available: Professional at £129 and Developer at £179. Neither version includes a resource editor, and although GEM is supported, none of the functions are documented, which seems more than a bit dumb. So figure on another £25 - £30 for a resource editor, and something similar for some good documentation. It makes probably the most expensive package available, and it also produces the best code (smallest and fastest) although Mark Williams C isn't far behind and Prospero is likely to be

near the top as well.

So, in conclusion! If you aren't sure whether you even like C, go for one of the public domain offerings, or GST-C. If cost is your main criteria then Prospero, Mark Williams or Lattice C are the ones to look for. If you are impatient then Megamax Laser C is the first choice. Finally, if quality of code matters then Aztec, Prospero and Mark Williams are your choices.

MicroEMACS 3.9 Release Notes by Daniel M. Lawrence

Available on volume 3 in the Source Library, this release adds some significant features

The distribution set is in three files:

EM39EDOC.ARC is the printable user guide
EM39EEXE.ARC includes the files:
README This document (These four files should be in your path for the standard setup)
EMACS.RC Standard startup file NEWPAGE.CMD Shifted Function key Pager PPAGE.CMD Programming page WPAGE.CMD Word processing page BPAGE.CMD Block and box manipulation page

ME110.RC	HP110 startup file
ME150.RC	HP150 startup file
AMIGA.RC	AMIGA ".emacsrc" startup file
ST520.RC	ATARI ST startup file
EMACS.HLP	Online help file
AZMAP.CMD	Translate AZTEC .SYM files to .MAP
BDATA.CMD	BASIC Data statement builder
FINDCOM.CMD	Find mismatched C comments
FUNC.CMD	Allow function keys on systems with non (like UNIX)
MENU.CMD	Sample Menu system
MENU1	Datafile for menu system
SHELL.CMD	Sample interactive MSDOS shell
TRAND.CMD	Generate random numbers and do statistics on them
MEIBM.EXE	EXE for IBM PC with mono/CGA/EGA

EM39ESRC.ARC includes the files:

ALINE.H	Atari ST graphic header file
ANSI.C	ANSI screen driver
BASIC.C	basic cursor movement

BIND.C	key binding commands
BUFFER.C	buffer manipulation commands
CRYPT.C	encryption functions
DOLOCK	file locking stub routines
DG10.C	Data General 10 screen driver
DISPLAY.C	main display driver
EBIND.H	binding list
EDEF.H	global variable declarations
EFUNC.H	function name list
EPATH.H	default path settings
ESTRUCT.H	configuration and structure definitions
EVAL.C	expression evaluator
EVAR.H	EMACS macro variable declarations
EXEC.C	macro execution functions
FILE.C	user file functions
FILEIO.C	low level file I/O driver
HP110.C	HP110 screen driver
HP150.C	HP150(A or C) screen driver
IBMPC.C	IBM-PC CGA and MONOCHROME driver
INPUT.C	low level user input driver
ISEARCH.C	incremental search functions
LINE.C	text line manipulation functions
LOCK.C	file locking functions
MAIN.C	argument parsing and command loop
RANDOM.C	other random user functions
REGION.C	region cut & paste functions
SEARCH.C	search and replace functions
SPAWN.C	OS interface driver
ST520.C	ATARI ST1040 screen driver
TCAP.C	Termcap screen driver
TERMIO.C	low level I/O driver
TIPC.C	TI-PC screen driver
VMSVT.C	VMS screen driver

VT52.C	VT52 screen driver
WINDOW.C	window manipulation functions
WORD.C	word manipulation functions
Z309.C	Zenith 100 PC series terminal driver

Recently, MicroSPELL 1.0 has been released. This program allows you to spell check text files and uses MicroEMACS to scan the file, doing various corrections.

MicroSCRIBE, a fairly SCRIBE compatible text formatter to go along with these programs will probably be available for beta testing early spring 1988. This program is turning out to be a lot more complex than I thought it would be, and is taking more time to get out.

I have in my possesion a port of MicroEMACS 3.8i to the Macintosh, and I will be incorporating the needed changes for the current version to support the Macintosh.

As before, I will continue to support these programs, and encourage everyone to spread them around as much as they can. If you make what you think are changes that are useful to many, send me the updates, and as time permits, I will incorporate the ones I understand, and agree with into the master sources.

MicroEMACS is available on disk directly from my by sending me $25 per order and a note specifying the disk format and the product that you need. I can fill orders for IBM-PC high/low density 5 1/4 and 3 1/5, ATARI ST single and double density, AMIGA disks and HP150 disks. (You do not need to send disks or mailers, I will provide these.) The distribution set includes on disk all docs, executables and sources. Also I will register you and you will receive automatic notices of new versions of all the programs I am releasing.

Commercial lisences to allow MicroEMACS to be incorporated into other software packages are also available at a reasonable per package price. Also I am available to do customization of MicroEMACS at an hourly rate. Send all requests to the address below:

USmail: Daniel Lawrence 617 New York St Lafayette, IN 47901
UUCP: pur-ee!j.cc.purdue.edu!nwd
ARPA: nwd@j.cc.purdue.edu
FIDO: The Programmer's Room 201/2 (317) 742-5533
ATT: (317) 742-5153

New Features since version 3.8i

New standard startup file

The new emacs.rc file is segmented into more parts and loads much faster than before. Separate "pages" of shifted function keys are available. Users can write their own "pages".

New Variables (there are a lot...)

$status	returns status of last command
$palette	color palette settings

$lastkey	returns last keystroke
$curchar	returns and set the ascii number of the character under the point
$progname	always returns "MicroEMACS"
$version	always returns the current version ("3.9")
$discmd	sets display of messages on the command line (except via the write-message command)
$disinp	sets echoing of characters during input on the command line
$wline	returns and sets # lines in current window
$cwline	returns and set current line within window
$target	returns/sets target for line moves
$search	returns/sets default search string
$replace	returns/sets default replace string
$match	returns last matched string in magic search
$cmode	returns/sets encoded mode of current buffer
$gmode	returns/sets encoded global mode (see appendix E in emacs.mss to decode this)
$tpause	returns/sets the pause for fence matching (this is in rather arbitrary units which WILL vary from machine to machine)
$line	return/sets the contents of the current line
$gflags	global operations flag (see emacs.txt Appendix G)
$rval	child process return value

New computer support

Atari 1040ST all three graphics modes and 50 line mode on a monochrome monitor. The mouse is bound to the cursor keys for now.

New Compiler support

Turbo C v1.0 under MSDOS is now a supported compiler. Mark Williams C on the Atari ST is also supported.

New directives

!while <condition>	loops while <cond> is true
!break	breaks out of the innermost !while
!endwhile	delimits the end of a !while loop

All !gotos are legal into and out of a !while loop.

Autosave mode

This mode saves the file out to disk every time 256 have been inserted. $asave controls the # of characters between saves, $acount controls the # of chars to the next save.

New functions

&and <log> <log>	Logical AND
&or <log> <log>	Logical OR
&len <str>	returns length of <str>
&lower <str>	lowercase <str>
&upper <str>	uppercase <str>
&rnd <int>	generate a random integer between 1 and <int>
&sindex <str1> <str2>	search for string 2 within string 1 returning its position, or zero if it fails
&env <str>	return value of DOS environment variable <str>
&bind <str>	returns the function name bound to the key <str>
&exist <str>	Does file <str> exist?
&find <str>	find file <str> along the PATH
&band <num> <num>	bitwise and
&bor <num> <num>	bitwise or
&bxor <num> <num>	bitwise xor
&bnot <num>	bitwise not
&xlate <str1> <str2> <str3>	scan <str1> replacing characters in <str2> with the coresponding characters in <str3>

Advanced word processing commands

^X^T	trim-line: trim all trailing whitespace
^X^E	entab-line: change all multiple char runs to tabs
^X^D	detab-line: change all tabs to multiple spaces

Merged EGA driver

The EGA driver is now part of the IBM-PC driver. This driver now supports MONO, CGA and EGA cards/modes. (settable by using the $sres variable)

8 bit characters fully supported

Eight bit characters (including foreign language and line drawing characters) are now supported on the various micro environments)

List Buffers expanded

Given a numeric argument, ^X^B (list-buffers) will now also list all the hidden internal buffers.

-k switch enhanced

If you use the -k (encrypted file) switch on the command line without a key immediatly following it, it will prompt you for the key to use to decrypt with.

word delete enhanced

with a zero (0) argument, M-D (delete-next-word) deletes the next word and not any intervening whitespace or special characters.

New File read hook

Whenever MicroEMACS reads a file from disk, right before it is read, whatever function is bound to M-FNR (which is an illegal keystroke) will execute. By default this would be (nop), but the standard emacs.rc binds this to a file that examines the file name and places the buffer int CMODE if the extension ends in a .c or .h. You can of course redefine this macro to taste.

Search Path modified

The order in which emacs looks for all .rc (startup) and .cmd (command macros) is as follows:

$HOME	(the HOME environment variable if it exists) the current directory
$PATH	(executable PATH) default list contained in epath.h

Line length limits removed

Lines of arbitrary length may be read, edited, and written.

Out of memory handling improved

EMACS will announce "OUT OF MEMORY" when it runs out of dynamic memory while reading files or inserting new text. It should then be safe to save buffers out IF THE CONTENTS OF THE BUFFER ARE COMPLETE at that time. When a buffer has been truncated while reading, a pound sign "#" will appear in the first position of the mode line. Also a # will appear in a buffer listing. If you attempt to save a truncated buffer, EMACS will ask if you are certain before allowing the truncated file to be written. As before, still beware of killing blocks of text after you have run out of memory.

DENSE mode on the Atari ST

On an Atari ST monochrome monitor, setting $sres to "DENSE" will result in a 50 line display.

Execute command

Execute-program (^X-$) will execute an external program without calling up an intervening shell is possible.

Better close braces in CMODE

The name says it all, try it.

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.

Pop Goes GEM!
By Phil Stubbington

Have you ever wondered what possible use hierarchical menus could be?

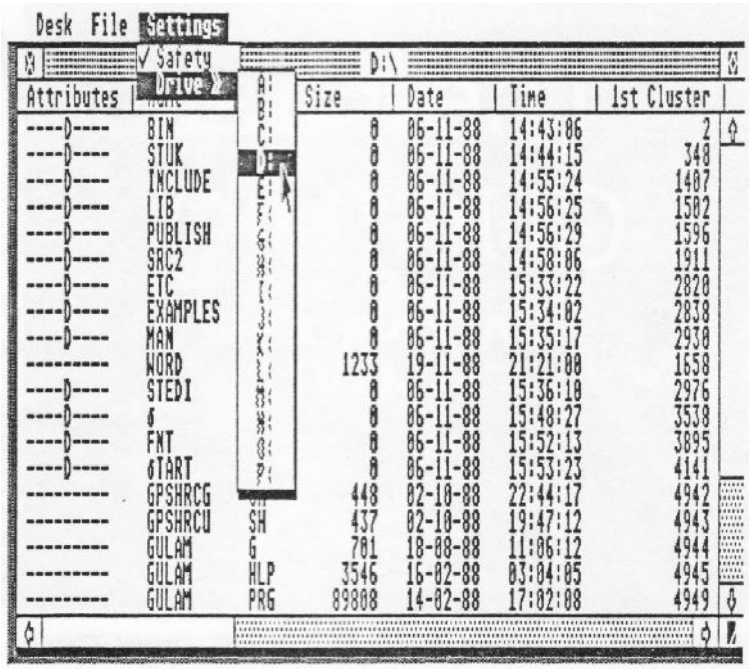
While putting together my latest masterpiece, DiskKit (short pause for shameless plug!), I realised that using a standard GEM dialog is not ideal in all circumstances. As part of DiskKit, the user can

select which drive the program will work on. Using dialogs in this instance is far from ideal. For one, after concentrating on the menu bar at the top of the screen, the user is then distracted to the centre of the screen where the dialog appears. Then of course, once the dialog is finished with, the user has to work out where they were before they were interrupted. Also, using an entire dialog for just selecting a single letter between A and P seems a bit like overkill!

As hierarchical pop-up menus are not a standard feature of GEM, and being a public spirited sort of person, listed below are the necessary routines from DiskKit for you to incorporate into your own programs. These routines are NOT public domain, they are strictly copyright. You must get written permission from me before using the routines in commercial (i.e. money making) software. In any case, the following message must appear in the documentation accompanying your program, and be displayed when the user selects the first item in the "Desk" menu (i.e. where the program credits are):

PopUp Menus: (C) Copyright 1988 Phil Stubbington

The routines are fairly well commented, so hopefully you shouldn't have too many problems using them. Note that these are only extracts, you'll have to write the rest of the program yourself! The routines have only been tested in medium resolution, so you may have to adjust the offsets slightly for other modes. Also you should note that (as with the standard GEM routines), we don't check if pop-ups go over the edge of the screen. If you're not careful this can cause exceptions (bombs). As it stands, the pop-up menus are created as standard dialogs. The entries are just ordinary STRINGs, with the SELECTABLE and EXIT fields enabled. One obvious improvement is to write a custom form_do routine, so that pop-ups operate in exactly the same fashion as real menus - items highlighted as the mouse passes over them, and clicking outside the menu de-selects it.



Hierarchical pop-up menus in action! Used in DiskKit to select a different drive, they can be useful in all sorts of applications, and even allow real WYSIWYG displays of typefaces and typestyles which standard GEM menus can't

And Finally...

Please let me know if you have any problems, or questions regarding this article and the routines. Similarly, if you have any other useful routines you would like to share with other CUG members, please send them in, with appropriate documentation.

The Code

```
#include <obdefs.h>
```

```
#include <gemdefs.h>

    :
    :

OBJECT *ObSecondPopUp;
int Dummy;

    :
    :
/* TSETTING and EDRIVE are from the #include file created by resource */
/* construction sets */
case TSETTING: /* Settings menu */
    switch(MenuEntry)    {    /* menu entry index */
        case EDRIVE:    /* select drive */
            Dummy = DoHierMenu(ObMenuBar,ObDriveForm,EDRIVE,test);
            ObDriveForm[Dummy].ob_state &= ~SELECTED;
            :
            :

/*=========================================================================*/
/* DoPopUpMenu                                                             */
/* Actions                                                                 */
/*      Re-selects whichever menu bar entry invoked us, and redraws it.    */
/*      Then draws the pop-up to the right of this menu bar entry, and     */
/*      (at present) invokes the standard form_do routine to handle        */
/*      interaction with the pop-up. Might be a good idea to write         */
/*      custom version of form_do!                                         */
/* Parameters                                                              */
/*      ObParent:   POINTER to OBJECT, must be the menu bar!               */
/*      ObChild:    POINTER to OBJECT, the pop-up menu/dialog              */
/*      Selection:  index to menu bar entry. EDRIVE in our example         */
/*      Baby:       POINTER to FUNCTION, which handles the pop-up          */
/* Returns                                                                 */
/*        ExitObject    index to exit object of _first_ pop-up             */
/*=========================================================================*/
int DoPopUpMenu(ObParent,ObChild,Selection,Baby)
OBJECT *ObParent;
OBJECT *ObChild;
int    Selection;
int    (*Baby) ();
{
int dx,dy,dw,dh;
int ExitObject;
int Ind;

    Ind = Selection;
    ObParent[Selection].ob_state = SELECTED;
    /* find the G_BOX type object which encloses the entire pull-down   */
    do  {
        if(ObParent[Ind].ob_next != -1)
   Ind = ObParent[Ind].ob_next;
    }while(ObParent[Ind].ob_type != G_BOX);
    /* now draw the pull-down - have to do this 'coz GEM automagically  */
    /* restores the desktop window before we get to look at it          */
    objc_draw(ObParent,Ind,MAX_DEPTH,0,0,400,400);
    /* our menu - which is really a diguised dialog, goes to the right  */
    /* of GEMs pull-down                                                */
    ObChild->ob_x = ObParent[Ind].ob_x + ObParent[Ind].ob_width - 1;
```

```
    ObChild->ob_y = ObParent[Selection].ob_y + ObParent->ob_y +
        ObParent[2].ob_height;
    dx = ObParent[Ind].ob_x-1;
    dy = ObParent[Ind].ob_y;
    dw = ObParent[Ind].ob_width  + ObChild->ob_width + 8;
    dh = ObParent[Ind].ob_height + ObChild->ob_height + 8;
    form_dial(FMD_START,0,0,0,0,dx,dy,dw,dh);
    objc_draw(ObChild,ROOT,MAX_DEPTH,dx,dy,dw,dh);
    /* interact with the first pop-up */
    ExitObject = form_do(ObChild,ROOT);
    /* might have another pop-up! */
    (*Baby)(ObChild,ExitObject);
    form_dial(FMD_FINISH,0,0,0,0,dx,dy,dw,dh);
    ObParent[Selection].ob_state = NORMAL;
    /* this will be the exitobject of the first pop-up! */
    return(ExitObject);
}

/*========================================================================*/
/* DoNothing                                                              */
/* Actions                                                                */
/*      Nothing! This routine _would_ be used if we had another pop-up    */
/*      hanging off the end of the first one                              */
/* Parameters                                                             */
/*      ObParent:   POINTER to OBJECT, must be the previous pop-up        */
/*      Selection:  index to pop-up entry, the actual drive letter in     */
/*      our example                                                       */
/* Returns                                                                */
/*      Nothing!    Perhaps it should return the exit object?             */
/*========================================================================*/
void DoNothing(ObParent,Selection)
OBJECT *ObParent;
int Selection;
{
}

IF IT WERE USED the function would run something like this:

int dx,dy,dw,dh;    /* clipping area */

    ObSecondPopUp->ob_x = ObParent->ob_x + ObParent->ob_width- 1;
    ObSecondPopUp->ob_y = ObParent->ob_y + ObParent[Selection].ob_y;
    dx = ObParent->ob_x-1;
    dy = ObParent->ob_y;
    dw = dx+ObSecondPopUp->ob_width;
    dh = dy+ObSecondPopUp->ob_height;
    form_dial(FMD_START,0,0,0,0,dx,dy,dw,dh);
    objc_draw(ObSecondPopUp,ROOT,MAX_DEPTH,dx,dy,dw,dh);
    Dummy=form_do(ObSecondPopUp,ROOT);
    ObSecondPopUp[Dummy].ob_state &= ~SELECTED;
    form_dial(FMD_FINISH,0,0,0,0,dx,dy,dw,dh);
```

The First CUG(UK) AGM

Ever get that feeling of deja-vu?

The Annual General Meeting of the C Users' Group (U.K.) has been re-scheduled. It will now take place on

SATURDAY, MARCH 4TH 1989

at the

ROYAL AERONAUTICAL SOCIETY, 4 HAMILTON PLACE, LONDON, W1.

The meeting will start at:

1:30pm

It is vital that members make an effort to attend, as without a formal committee the group is very limited in what it can do.

If you are going to attend, it would be appreciated if you could send us confirmation as soon as possible. Nearer the date, you will be sent a reminder and further details of the programme for the day.

WE WOULD PARTICULARLY like to hear from members interested in standing for committee posts. Also, if you are able to offer transport to other members living in your area, please send us details (meeting points, time, car registration, 'phone number, etc.) and a list will be distributed to all attendees.

YOUR CONFIRMATION just needs to say something like:

I wish to attend the 1989 Annual General Meeting of the C Users' Group (U.K.)

Membership #: _____

Signed _____

Please complete this form and mail it (or a photocopy) to:

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

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 5 DEADLINE

All material for issue 5 should be submitted to the usual address by 18th February, 1989.