The Magazine of the C Users' Group (U.K.)

Editorial

Welcome to the second issue of "C Vu", the magazine of the C Users' Group (UK). Since the last issue membership of the group has been going up in leaps and bounds, mainly as a result of a large plug in the user group section of Personal Computer World this month. By whatever route you came to join CUG(UK) you are very welcome!

As you are probably aware, CUG(UK) has two main aims - supporting the C language itself, and also other areas which relate to it. One of these "other areas" is, of course, operating systems. As if to prove the point, we take an in-depth look at the Mirage OS in this issue, and future issues will cover OS-9, Minix and distributed operating systems.

As with any publication, "C Vu" relies heavily on its readers for articles, reviews, etc. Many of you may be fairly new to C and going prematurely grey trying to get the hang of some of C's more idiosyncratic features. Don't just sit there, write to us with your problem and we will try to help. If enough people are interested, we will start a series on C for beginners, or perhaps the occasional article on certain areas (pointers being the obvious example!) This applies to any articles in "C Vu" as well, if you don't understand anything, write in and we'll attempt to explain it.

SPECIAL OFFER!

It's special offer time again! Last issue we brought you the MIX C system at under half price, this time we bring you the rather natty Discovery 1200P Pocket Modem, for under 110 pounds. Full details inside.

CUG(UK) is still in the early stages of development, so you may notice that things seem to change from one issue of "C Vu" to the next. The astute amongst you will have noticed the page numbering went a little awry in our first issue. So, to make sure this doesn't happen again, I've dispensed with page numbers altogether!

People have a wide variety of printers, font sizes, favourite paper sizes, etc. so future issues of "C Vu" (on disk or bulletin boards) will be in PROFF format (or something similar). When I have decided upon this, the necessary programs will appear in the CUG(UK) library, although as PROFF format is pure ASCII it is not essential to have the programs to be able to print out "C Vu".

THE NEXT ISSUE

Please get your letters, articles, and reviews to me AS SOON AS POSSIBLE. Preferably on disk, in straight ASCII (no margins please!). Mail can be sent via Martin Houston (address on the back page). If everything goes okay, the next issue of "C Vu" can be expected in late March.

Phil J Stubbington
Editor

CONTENTS

REGULARS:
o Editorial
A few words from the editor
o Letters Page
Commenting programs, how and why
o Everything You Wanted To Know About C .....
Tips and Tricks from Steven Palmer
o The Back Page
How to receive "C Vu" on disk, and a
membership form to get your friends interested!

SPECIAL OFFER:
o The Discovery 1200P pocket modem... 300/300 and
1200/1200 baud communications for under 110 pounds.
Brought to you by Digital Matrix Ltd.

SERIES:
o Structure, Part 2
*** Due to lack of space, this has been ***
*** held over until the next issue ***
Colin Masterson

ARTICLES:
o Get Online With Our Modem Offer
How to keep in touch by modem
Martin Houston
o The C Source Library
o A Wander Through The C Source Library
How the library is organised, what it will
cost you, and what's in it
Martin Houston
o Minix
How to get in touch with our Minix coordinator

REVIEWS:
o Mirage
A versatile operating system for Motorola systems
Phil J Stubbington
o C Programmers Guide To Serial Communications
From ASCII to XMODEM, via KERMIT!
Martin Houston

JOB OFFERS:
o News from Apricot

GET ONLINE WITH OUR MODEM OFFER

By Martin Houston

The CUG is now active on a number of Bulletin Board services (listed below). If you do not have a modem then you are missing out on the ability to download items from the library (for no charge!) and send in your comments in a form that can easily get into the magazine. We have negotiated a special offer on a modem that should suit the needs of many CUG members at a very keen price!

It's pocket sized, comes in its own carrying wallet, and should work with any micro equipped with an RS232 port, it's.....

- THE DISCOVERY 1200P POCKET MODEM!
- HAYES COMPATIBLE - AUTO DIAL - AUTO ANSWER!
- 300/300 AND 1200/1200, CCITT AND BELL COMPATIBLE!
- RUNS OFF A PP3 CALCULATOR BATTERY!
- SUB-MINIATURE CALL MONITORING LOUDSPEAKER!
- JUST 109.25 POUNDS TO CUG MEMBERS (5 POUNDS OFF)
- PRICE INCLUDES VAT AND CARRIAGE
- ONLY AVAILABLE FROM:
  DIGITAL MATRIX LTD.
  75 Willow Road,
  Solihull,
  West Midlands,
  B91 1UF
  Tel.: 021 704 1399.

This modem supports the more sensible V22 standard instead of the silly & antiquated V23 that many budget priced modems support. V23 is all very well if all you are doing is downloading large quantities of information but having to upload at a snails pace 75 baud means that V23 is a poor choice for anyone serious about a 2 way flow of information. A V22 modem will give a full 1200 baud speed in BOTH directions. The 300 baud V21 standard is provided for calling older, slower modems. West Midlands Opus & Dr Solomons Fido already support V22 operation and Chronos Lair is due to go V22 very shortly (and may have done so by the time you read this).

The cost of putting your machine 'online' is not as great as you might expect. Under 110 pounds for a Discovery 1200P, another few pounds for some communication software of your choice and the phone bill. What many people dread when they think about a modem is the huge increase in the phone bill! If you are sensible about modem use this is not as big a problem as expected.

The secret of economical modem use is to stick to off peak times. Calling a BBS in your own local call area is 5 pence for 5 minutes at cheap rate. This works out at only 60p an hour for a useful & entertaining passtime. the BBS is long distance over a 'low cost' route - such as calling a Birmingham number from London the charge is only 25p for 5 minutes or 3 pounds an hour. Normal long distance routes will cost 35p for 5 minutes or 4.20 an hour. Modem communications is an exciting and absorbing hobby & well worth getting into.

Digital Matrix also does an extensive range of other modems at higher prices - contact Clive Warner at Digital Matrix for more details.

LIST OF BBS KNOWN TO CUG

- Chronosoft Lair
  021 744 5561
  300N81 V21/23 at present
- Digital Matrix
  021 705 5187
  1200N81 V21/22
- West Midlands Opus
  021 711 1451
  1200N81 All speeds
- Dr Solomon's FIDO
  024 03 4946
  1200N81 All speeds
- Academics
  021 705 9677
  300N81 V21/23
- CIX
  01 399 5252
  All speeds

You will find a warm welcome from the sysop on all of these boards if you say you a CUG member. If anyone knows of any other BBS that have an interest in C I would like to know about them so that this list can be updated for the next edition of C Vu.

West Midlands Opus is well worth a call as it is participating in the international C programmers conference on ECHOMAIL. There is much interesting material in the conference, most of it is of U.S. origin. The board runs V21 through to V22bis with MNP error correction as well so if you have a high speed modem then downloading from this board can be very economical.

Dr Solomons FIDO is a very popular system & can be hard to get onto at times but always contains interesting technical material.

The best way to get into contact with me is to leave a message on one of these boards & I will reply. Any articles uploaded onto one of these boards can easily find its way into the next issue of C Vu.

Keeping things computer readable makes life easier for everyone!

Letters Page

A letter has flooded in from Colin Masterson, the author of our series on structured programming:-

I hope that readers might bear with me as I once more bang the drum about one of my 'hot topics' - that of commenting programs; let's be specific, commenting C programs.

As Mark Burgess seemed to be suggesting in a recent article in PCW, perhaps the time has come to try to standardise in some way a level of commenting which, together with structuring, will improve the useability, if I may use that word, of our software. As I see it, when we write a function - no matter how small - we have three clear obligations.

These may be summarised as follows:
1. Primary Obligation. To other users (& the author)

a) To provide sufficient information to allow anyone to call the function correctly and to make sense of any returned value.

b) To inform the user WHAT action is performed.

2. Secondary Obligation. To author (& further details for user)

a) To present a structural summary.

b) To note any peculiarities about the method employed such as recursion, calls to BIOS, DOS or I/O which may lead to portability restrictions.

3. Tertiary Obligation. (If function complex or part of suite)

a) Indicate caller function and/or related functions.

b) Indicate function level in the program hierarchy.

c) Indicate version, amendments and revision history.

d) Indicate any test or QA programs.

I shall propose a function header shortly which consists three parts corresponding to these three obligations, each part comprising a number of 'fields'.

Not all parts will be relevant in every case. The style and layout of the header may be left to the individual but some points may be worth noting. Consideration should be given to a standard form which would enable the use of utilities and filters to perform certain documentation and indexing tasks automatically. (eg through the use of GREP and the like.)

Since most people make use of a text editor with block copy and insert facilities, it makes sense to have a single dummy header and insert this into the source as required.

Furthermore, the NAME and CALL METHOD fields may often be directly copied from the function definition, avoiding repeated typing. (ED - an example of the header is given later on).

The need for information as noted under obligation 1 must surely be clear to everyone. For

complex functions the explanation of functions and parameters may be quite lengthy. For simple functions, very brief. The need for the following parts may be less obvious. In general though, it ought to be possible to fully understand the implications of using a function without having to study the code itself. Recursive functions or ones with a large quantity of local variables, place a heavy demand on the stack. This is worth noting.

If calls are made to routines such as interrupts, this has an effect on portability and should also be noted. For more detailed study, or more involved functions, a few lines of pseudo code and a note on peculiarities is always appreciated.

Accepted, it is often difficult to know what may be termed 'peculiar' in a C function. But if you shut your eyes and squint at the function, it doesn't take long to spot the parts for which "he'll wonder what on earth I'm doing that for" becomes an appropriate comment.

Finally, the hierarchy. This becomes very important once the number of functions breaks the 10 or 20 barrier; it's simply too much to remember what calls what.

I'll reiterate later my suggestions for levels (see recent letter in PCW) and use the adventure parser by Martin Houston as an example. (Absolutely no offense intended.)

Before I get called down for asking everyone to write a novel at the start of a three statement function let me at once suggest that these three obligations may in many cases be reduced to just one: the primary. It is always essential that we present the name and parameters to prospective users.

The function definition itself may sometimes be considered sufficiently clear to explain the action and calling requirements. I would argue however, that this is rarely the case. If our functions are created in a generalised form (as they should be to allow re-use) then the exact purpose of each parameter must be clearly stated.

For (what I call) Service Level functions at the bottom of the hierarchy, using only simple data types, then it may be quite plain what is required. But is it?

Consider the case:

```
int strcount(s,c)
char *s;
int c;
```

It is our responsibility to ourselves, and to other first time readers, to at least provide a minimum of explanation. At the very least, for these low level ones, this is acceptably done with brief comments around the definition:

```
int strcount(s,c)   /*  returns number of times 'c' is in 's'   */
    char *s;        /*  the string being scanned                */
    int c;          /*  char to count                           */
```

This should be sufficient for a user to make use of the function and to understand its returned value. We have fulfilled our obligations 1a and 1b, albeit somewhat briefly.

Notice that, even in this simple case, coded as:

```
int i;
     for(i = 0; *s; s++)
          i += (*s == c);
     return(i);
```

we have made no statement about boundary conditions - what if 'c' == NULL, if 's' is zero length, if no occurrences of 'c' are found, what maximum length for 's' ?

However, even fairly inexperienced users would probably be happy with the above comments.

The foregoing excusing from further commenting is on the grounds that the data types involved are simple, an array of characters being considered moderately simple. Were the function to involve parameters of structures, unions, more complex indirection or to manipulate global variables, then this simplification could no longer be considered valid. In such cases at least the NAME, CALL METHOD, GLOBALS and RETURN fields should be completed.

I don't deny that adding comments at the head of a function may take a few extra moments - but it is well worth it. In these days of 1M byte on a floppy and 40Mbyte hard discs, the space consumed by such comments is not worth mentioning. I am also aware that, in an eagerness to get something going, there will be a reluctance to add comments during early development. That is fair enough. But every now and then, before it's too late, take the time to go back through and add some little header to the start of the functions.

As an example here is a header for a function from a program I am currently working on:

```
/*-------------------------------------------------------------------------
   NAME         :   locate_item - locate next item in file.
   CALL METHOD  :
                    long locate_item(pge,fp)
                    int *pge;   address of present page number
                    FILE *fp;   file being read

   GLOBALS      :
                    end_of_file - set TRUE if EOF reached.
                    token       - used to hold next character.
                    limiter     - current item delimiter char.
   ASSUMPTIONS  :   pagestr has been set up with valid newpage sequence.
                    File is opened in BINARY mode.
   RETURNS/
   EXIT CODES   :   Offset in file of start of next item, 0L if EOF

   STRUCTURAL
   NOTES        :   Locates the next item in the file using the current
                    limiter char.  Check all the time for a possible
                    newpage code and keep a copy of it if it looks as
                    though it could be one.

                    Steps are:

                    While not EOF_CHAR
                        [
                        If token could be newpage then see if its last
```

```
                        one needed.  If so then bump page counter, else
                        just save token.
                        If token isn't a delimiter then we've found the
                        next item - return offset.
                        ] get next token
                        If EOF_CHAR then return error flag and set
                        end_of_file

   VERSION      :   1.01
   AMENDMENTS   :
   LEVEL        :   SELECT <- HANDLER <- CONTROL
   CALLER       :   sort_file <- main
-------------------------------------------------------------------------*/
```

Now, here are the hierarchical levels into which I divide my programs with reference to the code by Martin Houston in C VU.

CONTROL
PREPARATION
HANDLER
SELECT
DATA
SERVICE

CONTROL LEVEL:

Top level function, main() itself or one called by main(). If any I/O is carried out at this level then it will be to obtain high level command choices only, passing control to the next level (HANDLER). Initialisation is carried out by calling functions at the PREPARATION level. In the case of 'adshell': main() is the CONTROL level which calls the HANDLER level function parse(). It is likely that, in a real adventure, a PREPARATION level function would be called prior to the while() loop and that, possibly, the entire while loop and request for user input would be contained within a CONTROL level function which was called from main().

HANDLER LEVEL

Supervisor type level, parse() in this case is determining an action function to be called to actually do something. Like an executive in a multitasking system, HANDLER level functions may actually do little themselves but pass control to the functions which arrange for the work to be done.

SELECT LEVEL

help() and fexit() are SELECT level functions. They control the real work being done. At this level, the function will call an actual data gathering or issuing function to perform an action. SELECT level functions may be able to call a number of functions at the DATA level according to conditions at their entry time. In this case, printf() is a DATA level function, sending out actual data.

DATA LEVEL

Stand alone functions which accept fairly simple data types and carry out work under control of the

SELECT level. scanf(), printf() and your own particular input routines are examples of DATA level functions.

SERVICE LEVEL

The tools and dirty work level. Library functions (like strcount) are examples of this level. They operate on only fundamental data types and do not rely on global variables which are unique to a particular program. They should be entirely portable and form the 'glue' which allows DATA level (and possibly SELECT level) functions to do their job. I have said enough but look forward to comments from others.

Yours faithfully,
Colin Masterson.

Mirage

A Multi-User, Real-Time and Multi-Tasking
Operating System For The 680x0!

Atari ST Version Reviewed By Phil J Stubbington

Introduction

Considering its parentage, it is hardly surprising that C is so widely used in systems software. In the writing of an operating system, it is estimated that more than 70% could, and often is, written in C. However, no C compiler will ever produce code as efficient (in terms of size and speed) as a competent assembly language programmer.

In 1980, Swifte (pronounced Swifty) Computer Systems, a UK software house, started developing Mirage for what was then the new 68000 processor from Motorola Inc. Since then we have seen the 68010, 68020 and just beginning to appear is the 68030, all offering improvements over the previous version, whileretaining a high degree of code compatibility. The decision to write Mirage purely in assembly language has obviously paid off. Today, Mirage can be found in hundreds of commercial and educational sites throughout the world.

First Impressions

The version of Mirage I shall be looking at here is for the Atari ST (any model, although a mono monitor is essential). The main differences between this and any other version are the price (œ99 plus VAT) and the form it takes (Mirage is often provided on disc, but comes in ROM cartridge form for the ST). All versions of Mirage are largely object code compatible, and the disk formats are similar.

Mirage for the ST consists of an 128k ROM cartridge, a single-sided disk, and a user's manual. Once the cartridge is pushed into place (preferably with the machine off!) and with the boot disk in place, a warm-boot can be performed. Although the Mirage disk format is unlike GEMDOS (the

ST's native operating system), the first 19 blocks are left untouched, so that you can format a disk under GEMDOS (perhaps using one of the improved formatters, giving you 800k+ rather than 720k) and/or use some of these blocks for a bootstrap loader. The Mirage support disk contains a loader to transfer execution to the cartridge. By doing so, Mirage actually uses less memory than the ST usually does, as TOS (the collective name for GEMDOS, GEM, etc.), hasn't had a chance to initialise.

Mirage defaults to a single-user system, so you can start to use it without having to login in the conventional multi-user way. From here you have access to all the facilities of Mirage. In addition to the shell, kernel and filing system, as much of Mirage as possible has been packed into the ROM cartridge. Terminal, device and I/O board drivers and some of the more common utilities are included. This leaves more RAM for what you want to do, as well as alleviating the problem of errant programs corrupting the OS (the ST has minimal memory protection).

The Shell

The standard shell under Mirage will execute a command file called AUTOBOOT.CMDF, which is a convenient way of configuring the system. In addition to the execution of external commands, the shell has a sophisticated control language. To differentiate between internal and external the period is used, so the "if" statement takes the form .IF {expression} .THEN {action}. Other statements include support for subroutines, assignments, and terminal input (either a single character or a line) and output (enable and disable, write a line, and support for the Mirage terminal control functions.

Output redirection is also supported, using the usual ">" and ">>" to create/overwrite and append to an output file. The actual format is a little strange though: to redirect the output of a program called fred you would enter ">OUT FRED" rather than the Unix equivalent "FRED >OUT" ! Input redirection is not available in the same way, although you can do something similar from within command files.

In short, the Mirage shell offers many of the facilities found in Unix shells, although it is a much "cleaner" design, and probably faster.

The Kernel

Access to the multitude of system calls available under Mirage (a brief overview is given below) is through the "TRAP" opcode of the 680x0 family. For those not familiar with the processor, the TRAP opcode, accompanied by a 4-bit (ie. 0 to 15 inclusive) immediate operand puts the processor into supervisor mode, and executes an exception routine via a vector table. This table usually sits at the start of memory (it can't be moved on the 68000) - and on the ST is the only area of protected memory.

Brief Overview Of System Calls Available Under Mirage

- Exception Handling - in addition to the internal TRAP exception, the 68k family can respond to general errors like division by zero, attempting to address non-existent memory, etc.
- Interrupt Management - from I/O devices

- Global Communications Area - IPC (inter-process communication) and peripheral control
- Scheduling - the Mirage scheduler is priority driven: the 32 priority levels are split between real-time (preemptive and non-preemptive) and ordinary tasks. The priority can be adjusted and scheduling enabled and disabled with these functions
- Task Creation - creation and management
- Semaphore Operations - standard P & V
- Event Management - de/allocate, un/set and wait on
- Memory Modules - de/allocate local memory blocks. Also implements a simplified version of memory compaction, as found on Apple Macs & CDC Cyber mainframes. When possible, modules are moved down in memory so that they are contiguous.
- Bitmap Management - de/allocate bits from a bitmap
- Terminal Management - receive and transmit characters between terminals
- Terminal Functions - terminal-independent control
- File Management - creation, deletion, locking, etc.
- Date and Time - get, set and show in a variety of forms
- Command Line Parsing - switches, etc.
- Numeric Conversion - to/from ASCII, hex, decimal,etc.

As well as the usual system calls found on any multi-tasking OS, Mirage has a facility known as bolt-ons, which are a convenient way of adding system calls. These can be used for a variety of applications - standard bolt-ons include disk and sector caches, printer spooler, floating point support (either in software, or via co-processors) and a record management system called TRAP (which stands for TRansaction Processor, not to be confused with the TRAP opcode mentioned earlier).

Filing System

The filing system under Mirage can best be described as bush rather than tree structured. Instead of a root directory, with branches leading from it, Mirage has a potentially infinite number of directories at the same level. This is pretty much the same approach as VM/CMS, although I feel it is better implemented in Mirage. The reason for this approach is the large overhead caused by a true hierarchical filing system.

To move from directory to directory you use the CD (or LOG) command, which functions in much the same fashion as Unix. Typing CD on its own will tell you where you are, for example SYS0::DSC0:[UTIL], and CD ALPS will change to the directory of that name (assuming it exists, of course!). To list the contents of the current directory you enter DIR (with wildcards if required) and to list the directories on a device you would use DIRS (remember you don't have a root directory, so you can't enter DIR DSC0:)

The first part of the file specification (SYS0) is mainly for use in a networking or multi-processor environment. The SYS part is the name given to the node in the network and the 0 is the Node Slave Number (NSN). The NSN is useful in systems where you have multiple processors on the same node, and is mainly a result of system security.

The next part (DSC0) is the device name and drive number. Mirage is very flexible in names given to devices (and network nodes), so although we use DSC to indicate a disc drive, we could equally have called it FLP or HRD.

UTIL is the directory name (limited to four characters), and is one of the pre-defined directories under Mirage. For example, the "H" directory is where C header files are searched for, and BOLT is where bolt-ons reside.

As with most filing systems, it is seldom necessary to specify the entire path (you have the equivalent of a home directory), so the file specification is not as complicated as it looks! Filenames are 8 characters long, with a 4 character extension (although I can't really see the extra character in the extension making a vast amount of difference!)

In addition to drivers for a variety of network protocols, and standard Mirage disks, drivers are also available for ram disks, p-Systems, reading MS/PC-DOS disks (and standard GEMDOS format disks on the ST), and ANSI compatible tapes. This is by no means an exhaustive list - I suggest you contact Sahara Software (who market Mirage) if you have particular requirements. Of interest to ST owners, Sahara intend to have a CD-ROM driver as soon as they can get their hands on a unit!

Utilities

Over 100 utilities are provided with Mirage, so naturally I don't intend telling you about all of them! All the usual file-related utilities are available: rename, copy, delete, change ownership, etc. As with output redirection, the syntax is a little unusual: instead of "rename {oldname} {newname}" you would use "rename {newname}={oldname}".

Commands exist to kill, suspend and revive processes. Mirage has a number of very useful utilities relating to processes. You can, for example, set up a background task with a virtual terminal and switch between your real and virtual terminal(s) as required. It is also possible to insert characters into the input buffer of other processes.

EDIT, a screen-oriented text editor, also comes with the package. All the usual editor facilities are provided. EDIT also allows you to invoke a compiler/interpreter based on the file extension. EDIT is, however, only able to edit one file at a time, so it isn't well suited to modular languages (C, Modula 2, etc.)

On the memory management side, Mirage allows you to load modules (programs or data) into memory, and/or share them between different processes. When you attempt to read the module, it will be read from ram, rather than disk. This is obviously quicker, as well as being more versatile than a ram disk. Incidentally, this feature also appears in OS/2.

Language Support

Most of the popular languages are available under Mirage: including Pascal, Lisp, Fortran, BASIC and of course, C. All compiled languages, with the exception of C, compile to assembler source rather than straight machine code. This is then processed by the Mirage Assembly Language Programming System (ALPS). Unsurprisingly, since Mirage was written in assembly language, ALPS is a very powerful (and fast) system, complete with linker, librarian, source code debugger and profiler.

The two major languages under Mirage are assembler and Pascal. So what about C, I hear you cry? Most languages under Mirage are written in-house, and judging by the Pascal compiler provided for this review, are very well written.

However, with C, Swifte decided to use Lattice 3.03, via Metacomco of Bristol.

The main problems with Lattice are as follows:

- executable code is not re-entrant or shareable. This is partly the design of C itself, and also because Mirage uses the 68000 as its base processor, which has no built-in memory protection.
- execution speed is _very_ slow. For example, under Mirage, on an 8 Mhz 68000 ST, Lattice runs 652 Dhrystones/second. Using Megamax C under TOS the result is 1250 Dhrystones/second.

Attempts are being made to improve the situation with Lattice C, but it really is a great pity, considering that C is the ideal high-level language for system development, that this situation was not cleared up some time ago.

Conclusion

As I was reviewing Mirage, one thing became very apparent - a great deal of thought has gone into most aspects of its design and implementation. Obviously, being a fairly young operating system, Swifte had the chance to learn from the "mistakes" of other operating system designers. This is not to say that it hasn't faults, but both Sahara and Swifte are actively involved in the continual development of Mirage, so where possible these are being sorted out.

If you are an Atari ST owner, Mirage gives you an excellent (read _cheap_!) introduction to multi-tasking operating systems. If you have a serious application, perhaps in one of the areas I mentioned earlier, Mirage is certainly well worth a look.

Prices

For the Atari ST version, prices are as follows:-

MIRAGE OS	99 POUNDS
LATTICE C	129
Swifte-Pascal	99
Swifte-Basic	99
Swifte-LISP	59
Swifte-FORTRAN-77	129
ALPS	49

All prices exclude VAT and P&P, and are correct at press time. For further details, and pricing on other systems contact:-

Sahara Software Ltd.,
Unit 5-11 Bondway Business Centre,
69-71 The Bondway,
LONDON,
SW8 1SQ.
Tel.: 01-735-3806

Everything You Wanted To Know About C .....

By Steven W. Palmer

The following tips and tricks were collected together after a study of some potential problems encountered by beginners to C. Space does not permit me to include many other, similiarily useful, tips. If you have any to contribute, please send them to the editor.

1. Constants

All integer literal constants are stored as integer, unless you supply the L suffix. Beware of calling a function that expects a argument of type long using a literal. For example

```
lseek(fh, 0, SEEK_BEG);
```

may not work because lseek() expects the second argument, the offset, to be a long integer. It should be re-written as

```
lseek(fh, 0L, SEEK_BEG);
```

All floating-point literal constants are stored as double, rather than just plain float. So the following

```
float count;
...
count = 8.95;
```

will elicit a 'Data Conversion' warning from some compilers because the 8.95 has type double which has to be converted down to float. If you feel picky about minor warnings, you can get round this by casting the literal

```
count = (float)8.95;
```

Again, beware of calling functions with literals if they are expecting an argument of type float. All mathematical libraries tend to use arguments of type double as a sort of safety catch. You should also #include <math.h> so that the compiler can catch and warn about unintended conversions.

2. Operator Precedence

You have probably come across the case where assignments inside a conditional, such as

```
while ((c = getchar()) != ' ')
```

have to be bracketed, otherwise they will be interpreted by the compiler as

```
while (c = (getchar() != ' '))
```

Take some time to learn the C operator precedence levels.

There are some other catches for the unwary. For example

```
if (p & 0xC0 == 0xC0)
```

does NOT mean what it seems. The == operator has higher precedence than the &, so the expression must be re-written as

```
if ((p & 0xC0) == 0xC0)
```

A good C compiler should catch the former and warn the user about possible missing parenthesis.

3. #define macros

Except for simple macros, place the definition part of all #define preprocessor statements inside brackets. This avoids any ambiguity arising from apparently innocent usage such as the following

```
#define TABLE_BASE  0x2000
#define TABLE_LIMIT TABLE_BASE+0x1000
#define TABLE_SIZE  TABLE_LIMIT-TABLE_BASE/sizeof(long)
```

TABLE_SIZE is definitely NOT set to the correct value. The macro will actually be interpreted as

```
0x2000+0x1000-0x2000/4
```

(assuming sizeof(long) is 4 bytes), or 0x2800. The second and third #defines must be rewritten as

```
#define TABLE_LIMIT (TABLE_BASE+0x1000)
#define TABLE_SIZE  (TABLE_LIMIT-TABLE_BASE)/sizeof(long)
```

If the macro takes any arguments, always place the arguments inside the definition in brackets.

```
#define isdigit(c)  ((c) >= '0' && (c) <= '9')
```

Finally, never call a macro with an expression that modifies itself, or any other part of the expression in that same macro. For example, the following are very suspect

```
isdigit(*c++)
isdigit(*c += 1)
isdigit(getchar())
```

4. Type declarations

Complex type declarations, while they are fun to explore and show off in listings, are one of the biggest headaches in C maintainability. Beware that if you make use of complex declarations, your code may be difficult to port to another language, such as Pascal or Ada. However if this is not too much of an issue, you can sometimes create very sophiscated and efficent data structures this way. For example

```
typedef char (* fchr)[4];
fchr (*routine)(char *);
```

declares a pointer to a function that returns a pointer to a character array of four elements. It can be assigned an address by

```
fchr lookup(char *);
...
routine = lookup;
```

and called by

```
fchr cptr;
cptr = (*routine)("HELP")
```

On return, cptr will point to the start of a 4-element character array, and can be used to skip over those characters to the start of the next 4 characters using

```
++cptr;
```

If your compiler allows you to examine the generated object code in assembler form, you can get some idea of how much more efficent the object code is over a more straightforward approach.

5. Debugging

If you are unlucky enough not to be using a proper C development system which includes a source-code debugger, like C-trace or Codeview, you will probably be wasting time tracking down obscure bugs that a source-code debugger would track down within minutes. However, there are a few tricks that you can use to help debug your program without having to restort to a machine-code debugger. Here are some...

(1) Use the serial or printer port to route diagnostic messages. If you have a spare terminal, you can connect it to the serial port of your computer and direct all diagnostics so that they appear on the terminal. You can also use a printer, but be careful of placing the diagnostics inside large loops ... you can end up wasting a lot of good printer paper!

(2) Use assertions. Assertions allow you to check that a specific condition is true before you execute the next statement. If your compiler does not support assertions, you can use the following. Place it in a file called "assert.h" in your header directory.

```
#ifdef DEBUG
#define assert(n,e) {if (!(e)) { \
  fprintf(stderr,"Assertion number %d failed!\n", (n)); \
  exit(1); } }
#else
#define assert(n,e)
#endif
```

Use it like this

```
/* Switch on assertions */
#define DEBUG
#include <assert.h>
...
/* Check that x is non-zero */
assert(1,x != 0);
p = q/x;
...
```

If the assertion failed because x has the value 0, then the program will print the following and stop.

```
Assertion number 1 failed!
```

(3) Predefine all functions. By predefining functions, you are telling the compiler what types of arguments it expects and what type it returns. This

way, you will be warned if you accidently pass a floating-point value to a function that expects an integer, or try to use the result of a function that does not return a value. An example of a predefined function is

```
char *malloc(unsigned int);
```

This says that function malloc() takes one argument of type unsigned int, and returns a pointer to a character.

(4) LINT your source code, or use the highest possible warning level on your compiler. This ensures that the compiler will report on possible bugs that are normally ignored at the default warning level, on non-portable constructs or on wasteful code. LINT is a utility available under UNIX and some other operating systems that performs a very strict check of your source code.

(5) Use the C-Vu technical help section. If you are stuck with a really persistent bug, get in contact with either Martin Houston or Steven Palmer. If they can't help, then they'll do their best to find someone who can, or even contact the software house who developed your compiler to see if the bug could be a product of the compiler itself. That is what a user group is for!

C PROGRAMMERS GUIDE TO SERIAL COMMUNICATIONS

by Joe Campbell. Howard W. Sams & Co. ISBN:0-672-22584-0 $22.95

Review by Martin Houston.

As the title suggests this is a book for anybody interested in serial (RS232) communications from a programmers or technically literate users point of view. The book starts with actually explaining fundamental principles of serial communication such as the ASCII character set and principles of serial communication. Joe Campbell has a great knowledge of his subject and presents reasons why things are the way they are rather than just dry facts. This approach makes even 'dull' parts of the subject interesting reading.

This 655 page book covers just about everything anyone would need to know about RS232 serial communications on microcomputers. I found the history of serial technology fascinating. The whole business of MARK, SPACE, START & STOP is made much clearer when the historical development of the concepts is properly traced (and it is historical - serial communications is based on ideas little different from 100 years ago!).

The Xmodem & Kermit protocols are explained and special attention is given to explaining error checking such as CRC. The book covers what the pins on an RS232 interface actualy do and fully documents the entire Hayes Smartmodem command set.

On the C programming side the book concentrates on the design & implementation of a terminal emulation & file transfer program. The program is designed to be portable - communicating with the serial hardware through a 'virtual UART'. A listing of the complete program suitable for the IBM PC is provided in an Appendix of the book.

In conclusion this book is a mine of very useful information about serial communications & how to program them. It is very readable and a worthy

addition to any C programmers library.

THE MINIX OPERATING SYSTEM

Much of C programming is influenced by the UNIX operating system and its derevatives. UNIX was the first operating system to be written in C; indeed it was the reason C was invented!

MINIX is a re-write of the UNIX operating system in C that does not contain any source code belonging to AT&T and therefore does not require a (very expensive) UNIX licence to run.

Don Forbes has agreed to act as co-ordinator for interest in MINIX within the C User Group. The next issue of C Vu will have a page dedicated to MINIX devotees.

If you run a MINIX system, are thinking of doing so or would just like to know more then Don can be contacted at:
Don Forbes, 35 Upland Road, South Croydon, Surrey, CR2 6RE.

THE CUG(UK) SOURCE LIBRARY

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 Archimedes 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 ST and most of IBM compatible portables. Single sided variants of the above formats are also available if requested.

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 have a directory to itself. 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. 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.

If you cannot cope with an ARCed file then a kind message to the SYSOP of the board will 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 useable by any machine (with adaption 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 benifit 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 usually 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 seperate totaly 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 compatable hardware will be isolated from generaly 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 endavoured 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 at the following rates (all fully inclusive):

COST (Inclusive)	
Your own 5 1/4" disk standard volume	2.00
Our 5 1/4" disk standard volume	3.00
Your 3 1/2" disk standard volume	2.00 *
Our 3 1/2" disk standard volume	3.50
Specially selected material	4.00 surcharge on above rate.

* A 3 1/2" disk will hold two standard volumes if formatted as double sided (this does not suit some machines). If you want a second volume on the same disk then the charge is only 1.00 extra.

PLEASE NOTE that if you want a single sided 3 1/2" disk then please say so when you order. The 3 1/2" 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 520 ST's come with a single sided drive 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. Hopefuly 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 effectivley 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.

A wander through the C library.

by Martin Houston the CUG Librarian.

The C source Code library is in its early days at the moment but already there is some interesting material that has been submitted. In this article I will draw your attention to some of the things new in the library that you may like to try out. I do not claim to have any extensive knowledge of any of the programs that I mention below. What we need is for people who have put in the effort to understanding something about specific programs, such as the EMACS editor, to share that knowledge by writing a review for C Vu about it.

Well here goes with some of the things that I think may interest you in the initial library offerings.

Library volume 1 is a good place to start. It's a collection of software tools to aid in the job of writing other C programs. What follows is an annotated form of what the MS-DOS DIR command lists the disk as. The number is the size in bytes of the file.

```
CCHK     C       4356  A bracket & comment checker - very simple 'lint' type
program.

FAULTY   C        104  A small faulty C program to demonstrate cchk
ABOUT    C       3667  A help lookup program
ABOUT    DAT     7424  with a database of C function call usage.
CFLOW    C      26750  This is a C program structure analyser
CFLOWX   C      25084  extended features for cflow.
CFLOWX   H       2688  required header for cflow & cflowx
CFLOW    DOC     2944  Docs for cflow & cflowx.
XC       C      20149  Three variants of the xc C cross referencer
XC1      C      22483  These were once the same program but got here
XC2      C      27095  by different routes!
XC       DOC     1536  Docs for xc.
DUMP     C       2877  A simple
DUMP2    C       6412  and a more lavish hex dump program.
MEMCLEAN DOC     3712  a program for cleaning memory on a PC.
MEMCLEAN C       6096
SNAP     C       2987  takes a memory snapshot for debug purposes.
ZAPLOAD  C      13349  A hex loader - see document.
ZAPLOAD  DOC     8832
HD       C       2439  Hex dump of a file.
```

```
HDR      C       3305  Displays .exe file header info.
TIMER    C       1778  timed execution of a command.
WHAT     C       2120  finds and prints module id strings.
HEXBIN   C       5721  converts hex dump to binary file.
HEXCNV   C       4961  converts hex dump to rom images.
```

As you can see this is all material of use to any C programmer (if you don't get similar tools with your system anyway). If you are keen on the UNIX system and the sort of software tools available under it the Library disk 6 will be of interest. It contains public domain versions of three important UNIX system utilities. First is a version of the Make program maintainer. Many C compilers come with a 'make' program nowadays but if yours does not or you are just curious about how make works then this will be useful.

Also on the disk is a version of the UNIX 'AWK' pattern matching language called BAWK. The AWK language has been called a '4th generation language' and is very good at jobs involving complex text manipulations. The third major program on disk 6 is a version of the UNIX 'stream editor' sed. In some respects this program is similar to BAWK. Its use is for non- interactive editing tasks. The program comes with a useful example 'sed script' for converting a Pascal source file into C. Disk 6 also contains several other minor UNIX style utilities and is certainly well deserving of further comment in future issues of C Vu!

Library volume 8 is another area of interest to people interested in UNIX. It contains not one but two attempts to provide a UNIX like 'shell' under MS-DOS. Each of these is a lot of source code for common UNIX style utilities. Sadly neither 'shell' can cope with multi tasking background processes but otherwise both give a 'feel' of a real UNIX shell. The programs seem to be quite MS-DOS specific but may be adaptable to other machines - why not give it a try!

Library volume 11 deals with the twin subjects of serial communications & library management. It contains several comms protocols (kermit & SEAlink) and several library managers including ARC, LUMP and SQUEEZE.

```
SEALINK  C      20716   The SEAlink comms protocol functions
ARCSRC   <DIR>          Source for the ARC archiver
LAR      C      16963   LU format library manager
LUMP     C       1680   LUMPS several files into 1
UNLUMP   C       2017   Undoes the effect of lump
SQ       C      22142   Squeezes (compresses) files
TYPESQ   C       6281   Types a squeezed file
USQ      C       6785   Unsquezes a file
LDIR     C      10792   Gives a directory of a .LBR library
LDIR     DOC      325
LTYPE    C       3809   Types a member of a .LBR library
BSPLIT   C       3607   Splits one big file into many small ones
BCOMBINE C       2813   Combines many small files into one big one
PC_COM   ASM    14464   Low lev support for IBM PC serial comms
CASYNC   ASM    10544   Low lev support for IBM PC serial comms
IBMTTY   C      16691   Terminal program for IBM PC
COMM     MNU      640
CRC      C       6374   Functions for CRC calculation.
KERMITPC C      31882   Kermit program for IBM PC
KERMITPC HLP     4864
```

Last in our short tour through the library is **volume 14** This volume contains sources of the UNIX Dungeons & Dragons game of Larn. Files are supplied SQUEEZED to fit onto the volume and the source of squeeze & unsqueeze is provided. This really is a jam packed volume, the Larn game should work under UNIX systems and

also MS-DOS. There should be few problems in getting it to work on other machines though. One word of warning - it needs lots of memory to run!

Contents of CUG Library volume 14

```
SQ       C      22142   Squeeze program that prepared these squeezed files
USQ      C       6756   Program needed to UNSQUEEZE the larn files
README            741   Instructions for getting larn off the disk
UNPACK   BAT       421   Batch file for unsqueezing all larn sources
LARN     LQK       385   Instructions to MS-DOS linker for larn
LTERMCAP LQB      4858   Library needed for DOS larn version

The following are the larn source files:
BILL     CQ       4199 CONFIG   CQ       1585 CREATE   CQ       9828
DATA     CQ      19305 DIAG     CQ       7583 DISPLAY  CQ       8991
FORTUNE  CQ       1608 GLOBAL   CQ      11575 HEADER   HQ       9352
HELP     CQ       1763 IO       CQ      16016 IO       IQ       9460
MAIN     CQ      18133 MAIN     IQ      10321 MONSTER  CQ      29332
MOREOBJ  CQ       6411 MOVEM    CQ       7529 NAP      CQ       2060
OBJECT   CQ      18384 PWD      HQ       1168 REGEN    CQ       2783
SAVELEV  CQ       1373 SCORES   CQ      14987 SEX      LQ        329
SGTTY    HQ       1880 SIGNAL   CQ       3872 STORE    CQ      15053
TOK      CQ       4013
```

Structure, Part 2

By Colin Masterson

In the last part we spoke about structure in terms of ignoring detail. We now go on to formalise our thinking.

Identifying structure

We can see from what's been said that it is not sufficient just to use a structured language to ensure a structured result. We must apply some thinking to the layout too. (Interestingly, it's just as easy to use languages like BASIC to produce structured code, once you understand the principles.)

We mean two things by structure.

1. Logical
   That is, the logical sequence of events that a program takes.
   Task D follows task C follows task B follows task A.

2. Presentation
   That is, the appearance of the printed statements for each task on the page. Page headings, indenting, use of variables names and whitespace fall into this category.

These two are completely independent. It would be possible to present a program :

```
Task E
Task A  Task B
Task D
    Task C
```

which was completely structured from the logical point of view; A calls B. B calls C and C calls D and E. But its use of variable names, indentation and layout give no clue as to the flow or structure.

Similarly, we could have a beautifully presented program:

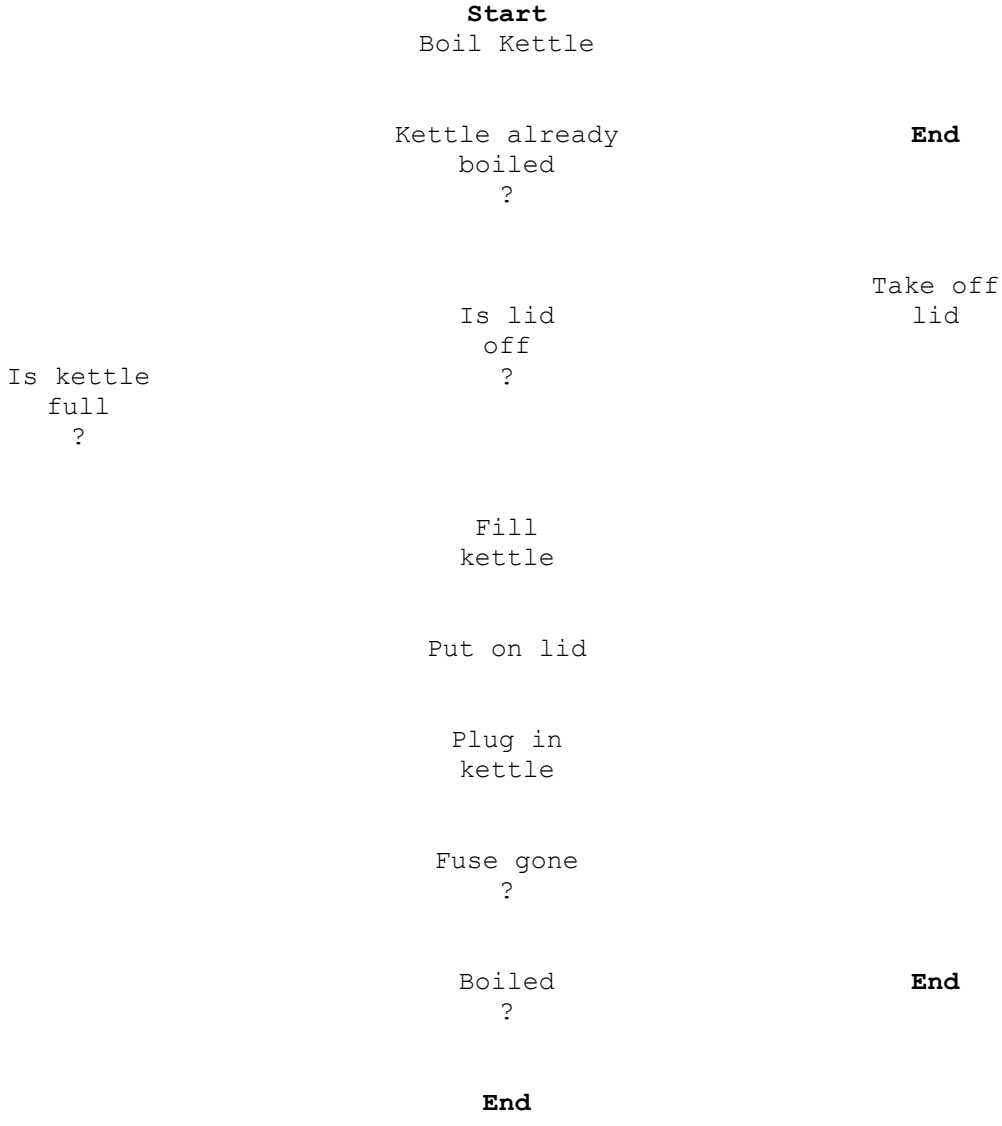
```
/*  this is a nice heading  */
Task boil_kettle
    Task Fill_kettle
Task boil_kettle
    Task fill_teapot
Task make_tea
    Task fill_teapot
```

In this case the program logic is not present.

In a strict definition of the phrase, "structured programming" we mean logical structure. I would suggest however, that the two should be considered so closely related as to be inseparable.

Let's assume therefore, that by structure we mean; a logical breakdown of a program into hierarchical levels, where each level is presented in such as way that this logical structure is clear. We can now move on to consider more formally what structure may be taken to mean.

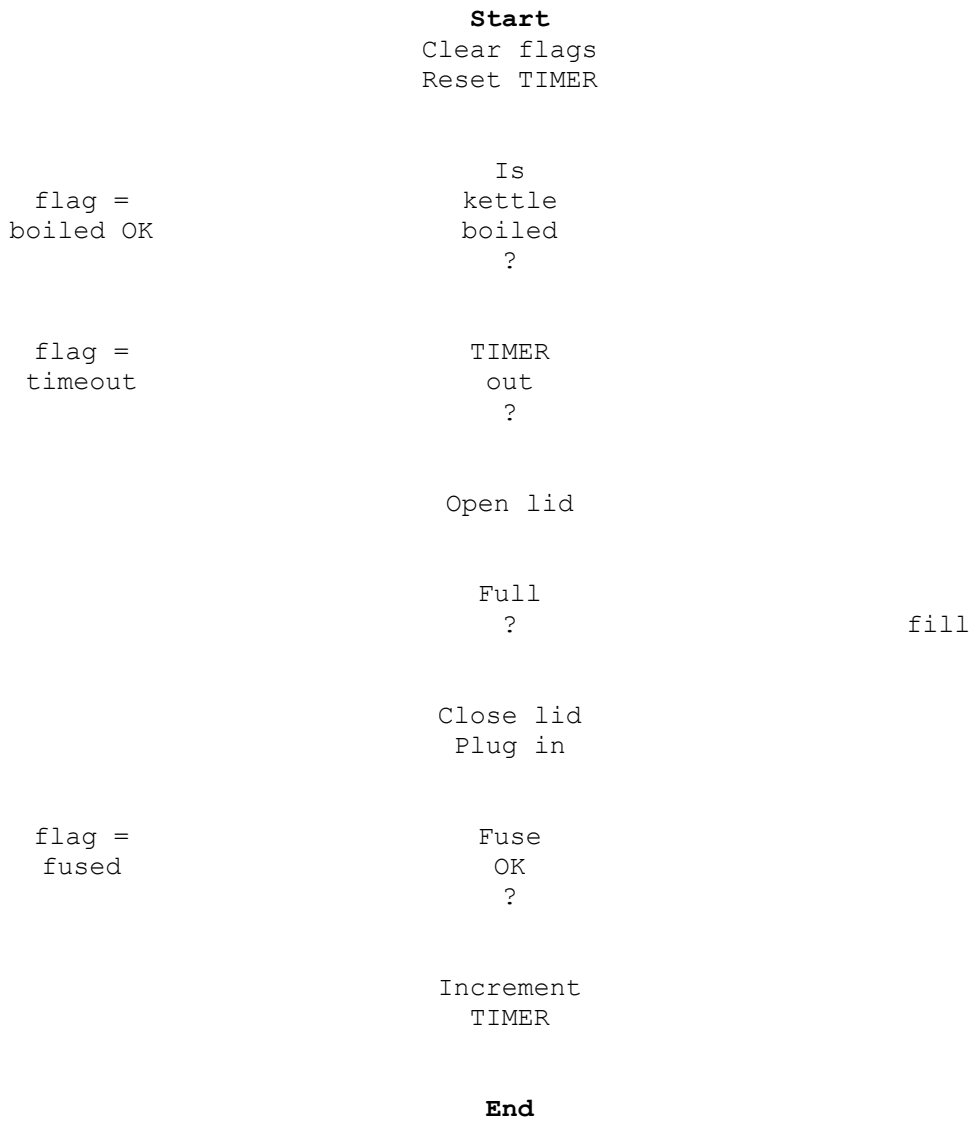
Let's look at our example again.



Not a very structured looking flow chart, but why not? Everything is at the right level, we've asked all the right questions. True - but another golden rule for structured programs is so called single entry - single exit routines.

Why? Well for a start, consider trying to find out why this function wasn't working. Let's say it's ending at 'blown fuse' all the time and we don't know why. How has it got there ? Or consider a case where operation seems to stick at 'fill kettle'. There are an infinite number of times we might have looped around testing the lid and full conditions and no way of knowing how many, or how it eventually falls out at 'is lid off'.

Now consider this flow diagram for the same thing:



In this case the task has a single entry and a single exit. Considering sub parts, each sub part also has a single entry and single exit. Only in the case of the first decision box does this appear to have two entry possibilities. The rule in this case is that such a situation would be catered for by a loop instruction. eg for(), do, while etc. (Never jump back with a goto !)

Translated to 'C', this might become a while construct.

```
    eg      while(kettle not boiled)
                {
                :
                :
                }   /*  end of while    */
```

Finding out what is going on is now quite straightforward. Now, if we are getting stuck at filling the kettle there is only one possible path which got us

there.

Further, even although there may be no final use for it, including a variable such as 'timer' can be very useful. We can use this to check and monitor the progress of a loop and, as in this case, to detect an error condition. Notice that the use of a for() loop allows this variable to be handled and also provide loop termination.

```
eg      for(timer = 0; timer < TOO_LONG; timer++)
            {
            if(kettle == boiled)
                {
                flag = OK;
                break;
                }
            :
            :
```

Identifying, or coding, single entry and single exit tasks can become quite tricky with a more complex situation. It becomes easier with practice and familiarity with the language, however.

If loops are becoming nested and intertwined to such a degree that their purpose is becoming obscure then the time has come to create another level. In languages such as 'C', this further subdivision causes very little overhead resulting in very little increase in execution time.

Let's recap:-

To produce a structured program we must:-

- Concentrate initially on high level tasks.
- Recognise special cases.
- Produce simple, single entry/exit processes.

In the next part we'll look at moving from the problem being considered to starting to produce some code.

Research & Development Division.
90 Vincent Drive
Edgbaston
Birmingham B15 2SP

tel 021 427 3002
fax 021 471 2935
telex 334754

C USERS' GROUP (U.K.)

CUG(UK) is a special interest group for everyone interested in the "C" programming language and related topics such as compilers & operating system design. It attracts programmers of a wide range of abilities from beginners to professional systems programmers.

The main activity of the group is the production of a magazine "C Vu" (membership covers six issues) which contains articles of interest to the C programming community.

An extensive C source code library is also available for the use of members.

Please complete the form below and send it with a cheque or postal order for ten pounds, made payable to C Users' Group (U.K.) to:-

Martin Houston, 36 Whetstone Close, Farquhar Road,
Edgbaston, Birmingham B15 2QN.

Name: _____ Address: _____

_____

Hardware Interests (machines used):_____

Software Interests (which C compiler used etc):_____

If you are already a member, but wish to change the way you receive "C Vu", please enter your membership number here: (_____)

Preference for magazine (please tick):

( ) Paper copy
( ) Standard 360k PC format 5 1/4" disk
( ) Single sided 3 1/2" disk (suitable for Atari ST, PS/2, & Apricot)

NOTE Due to the higher cost of 3 1/2" disks, at present the group will require a surcharge of 50 pence per issue to get the magazine in this way. Hopefully, the need for this surcharge will disappear

Please do not pay the surcharge in advance. If you select 3 1/2" disks each issue of the magazine will tell you if the surcharge is still in force. Please then send the 50 pence or return the disk ready for the next issue of the magazine.