October 1987 C Vu Volume 1, Number 1

Editorial

Welcome to the first issue of "C Vu", the magazine of the C Users' Group (UK). Before we go any further, I would like to say a very big thank you to all those people who have written for the magazine. Please keep it up! Details of how to get in touch with the group are to be found on the back page.

*** SPECIAL OFFER TIME! ***
*** SAVE UP TO 40 POUNDS! ***

Our special HALF PRICE offer in our premier issue is from Analytical Engines Ltd., who are the UK distributors of the MIX C. The MIX C system was reviewed by Nick Walker for Personal Computer World this September. He said the symbolic debugger Ctrace (part of this offer) was "by far the most delightful debugger.... I actually found myself enjoying debugging!". "MIX C has a truly excellent tutorial which makes it probably the best buy for an absolute C novice"

Full details of the offer can be found on page 7.

Using C for adventure writing seems a very popular subject this issue, with two articles, one about ADVSYS from Martyn Dryden, and ADSHELL (an adventure shell) from Martin Houston. The source code to ADSHELL is presented here, and both are available from the CUG software library.

We also take a look, with Colin Masterton, at how to structure your C programs. Believe me, it really is worth the extra effort in the long run!

*** 1986 Members ***
*** Please see page 27 ***

If you were a member of the original CUG(UK) you will know it had a somewhat shaky start, but we are now (as they say) "under new management". The C language is becoming increasingly popular, and CUG will support all levels of user, from proficient hacker (in the original sense of the word) to bemused beginner.

That's just about it from me, this time around (pause for communal sigh of relief!). Please keep the magazine material coming, and see if you can get a friend to join CUG(UK) before the next issue!

Happy Coding,
Phil J Stubbington

CONTENTS

All About CUG(UK)
Aims and objectives
By Martin Houston

The C Users Group (UK) is an informal group for all people who have an interest in the C language and related topics such as the design of 'systems' type software such as operating systems, language compilers and other types of work usually done in C. It is hoped that the Group will attract a wide range of abilities and backgrounds, from the home user experimenting with C purely as a hobby to professional software engineers using C in major work.

The group is NOT an 'IBM PC & Close compatibles only' group. The whole point of C is that it frees the programmer from the shackles of working with a single type of processor. It is true that the vast majority of CUG members run their C programs on PC clones but the watchword of the group must be portability.

Atari ST owners come a close second and we would like to see a free interchange of software between all machines. If it is done properly then C software developed on an 8088 machine today can be just as relevant to an Atari ST, Amiga or an Archimedes or even to some processor that has not been invented yet.

The current activities of the group are:

1. "C Vu Magazine"
This magazine, to be published whenever there is enough material and money in the bank to pay for its printing. The economics and therefore frequency of magazine production will depend on the number of paid up CUG members the group can attract (the per-unit cost for printing a thousand newsletters is MUCH less than only printing 100).

If each existing CUG member could get two or three friends with an interest in C to join then the groups position could be greatly improved. The 10 pound subscription covers the production cost of "C Vu" and the general admin costs of forming the group. The C source library is seperately financing by charges made for withdrawal of material.

2. Dial In Bulletin Boards
Based in Birmingham and Guildford, Surrey these boards will have downloadable copies of every issue of "C Vu" and some of the CUG software library. Both boards are accessible to anyone with the appropriate modem & software. Please see page 28 for more info.

3. Source Code Library

A library of member contributed public domain software. The library WILL be in source code form only. You may expect to do some work in getting material from the library to work on your machine (that's one of the reasons we have a problem page. ED) but that is all part of the fun and the learning process in porting software.

The preferred method of access to the library is via the boards as no problem with disk formats or postage/admin is involved. For postal reasons the library will be organised into logical units of up to 360k. If you are requesting material on 3.5" media then two library disks will fit onto one disk. If you are using the boards long distance then it will be much cheaper to just browse while online and order the disks you require by post:- downloading 360k can play havoc with your phone bill!

The charges for using the library are as follows:

DONATIONS: No charge.

If on making a donation you wish to have your disk returned with other library material on it then just include one pound for return P&P.

WITHDRAWALS: 3 pounds to 5 pounds per disk.

If you supply the disk(s) for the library withdrawal then the charge is three pounds per disk. For 720k disks the charge applies to each 360k of data they contain. Thus a full 720k disk would be charged six pounds. Although 720k disks are easier to post making up the disks from the library is harder in the first place. If you wish the library to supply a disk for the software then an extra one pound for 360k and two pounds for 720k is payable (note that this saves you postage on the outward journey).

An up to date library list will be kept on the boards and is also available from me. Remember that the library is only as good as the members make it. If you manage to make improvements to anything from the library please send the improved version back when you are happy with it. It would be nice to get variants of the more complex programs that have had the bugs hammered out for all the popular machines.

Finally, Sinclair QL and Amstrad 8-bit users. Both these systems have non standard disk systems. For QL owners there is a program available from Quanta (the QL users' group) that will allow them to read standard MS-DOS format disks. For Amstrad and other machines with the 3" drive is there anyone out there willing to be 3" drive sub librarian? Is there an enterprising Amstrad owner out there willing to invest in a modem so that he can download the CUG library onto 3" disks for Amstrad owners?

These things are hopefully just the start. What is needed now is for you founder members of CUG to get out and get the group expanding from the small enthusiastic nucleus that we have at present. The range of activities that we can organise is limitless but all require people willing to to them.

ED - an abbreviated version of this article will appear in all future versions of "C Vu", to keep existing members abreast of any new developments, and to let new members know what the group is all about.

/****************************/

Writers Start Here

By Phil J Stubbington

There are no hard and fast rules about writing for "C Vu", but the easier you make my job, the more chance you stand of seeing yourself in print! Basically, all submissions should be in straight ASCII format, unless you have an Atari ST, in which case STWRITER format would be preferred.

I see my role as editor as twofold, firstly, to massage your text into a suitable format -- cutting and adding bits as I go --, and secondly, correcting spelling and typographical errors (and, naturally, adding a few of my own!).

Submissions on paper, unless they are short (ie. one side of A4), are unlikely to be published. The ideal submission would consist of a brief outline (printed out, preferably) with the actual text on disk, or if you are into communications, transmitted to "CUG(UK) Online" (details on the back page).

If you wish to include diagrams, please get them printed out if possible. You could even go really lo-tech, and get a ruler and pencil out!

/****************************/

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

Problems? Can't figure out why your program doesn't do what it is supposed to do? Troubles porting software? This is what this section of the magazine is all about. We currently have two members who can offer help on a variety of hardware, software, and operating systems. If you should have any particular expertise, get in touch.

MS-DOS, Atari ST's, OS-9.

For help with any of the above, please write to:-

Steven W Palmer,
L. J. Electronics Ltd,
Francis Way,
Bowthorpe Industrial Estate,
Norwich,
NORFOLK,
NR5 9JA.

Unix, Xenix

Problems with C running under these OS's, & anything else!:-

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

In either case, please mark your envelope "CUG - Technical Query". Personal replies cannot be guaranteed, but you can always send a stamped, addressed envelope just in case!

CUG SPECIAL OFFER

The MIX C compiler & ctrace debugger as reviewed in September 'Personal Computer World' magazine are available to CUG members at the following special prices:

MIX C Compiler	19.95
MIX Split screen text editor	19.95
MIX Ctrace (PC only)	19.95
MIX ASM Utility	6.95
MIX C Examples disk	5.95
Complete MIX C-Works (PC only) (includes ALL of the above)	39.95

Apart from Ctrace (which requires an IBM PC or compatable) the MIX C system is suitable for all MS-DOS machines. CP/M versions of MIX C are also available.

This offer is valid until 31st of October 1987. If you wish to take advantage of it then write to:

David Hurley
Analytical Engines Ltd.,
PO BOX 35,
Eastleigh,
Hampshire,
SO5 5WU

Please quote your CUG membership number to qualify for the special prices.

/****************************/

An introduction to ADVSYS
by Martyn Dryden

ADVSYS is an adventure writing system - a language compiler and interpreter dedicated to writing text adventure game programs. It was written by David M Betz of Manchester, NH, and placed in the public domain for unrestricted non-commercial use.

The system consists of compiler and interpreter programs. The compiler operates on your game source code to produce a data file. This is a packed binary file at which the game user can't usefully peek. Your game consists of the data file, plus the interpreter program with which the user `plays' it.

ADVSYS is written in C. The portability of C is crucial to ADVSYS, which itself takes portability a stage further. Because ADVSYS can work on any computer for which a C compiler is available, a game produced using ADVSYS can be played on any such computer. The game source code simply needs to be run through the ADVSYS compiler on the target machine, and then teamed up with the ADVSYS interpreter for that machine. As well as making it possible to write a program that will run on virtually any computer, ADVSYS relieves the game programmer of all low-level instructions and machine dependencies.

Programmers can put maximum effort into the creative aspects of adventure programming: the objects, locations, and actors, and the way they interact. A further benefit of ADVSYS is that it provides game authors with a standardised framework for them to communicate on aspects of game programming. This could encourage people to try their hand at writing an adventure and may possibly lead to the growth of public domain adventures and co-operative efforts.

Sample ADVSYS code

To give you a quick overview of the ADVSYS language, here's a typical fragment of game source code:

```
(location storage-room
(property
     description "You are in a small storage room
                  with many empty shelves.  The only
                  exit is a door to the west."
     short-description "You are in the storage room."
     west hallway)
(method (leave obj dir)
     (if (send obj carrying? rusty-key)
          (send-super leave obj dir)
          (print "You seem to be missing
                  something!\n"))))
```

This code defines an object of class location, called storage-room. (Assume that we have already defined what a location is.) Its properties include long and short descriptions and the direction of the adjacent location.

It also has a method, a routine to be executed whenever the system sends that location a particular message. In this case the message is leave.

Elsewhere in the program we will have defined under what circumstances the message leave will be sent to a location. Here we have to define what happens when this location receives that message. We want to prevent the actor leaving the location without carrying a particular object, the rusty key.

The message brings with it two parameters: the identity of the object that is leaving (this will be an object of class actor), and the direction of travel. Because this location has no directional properties other than west, ADVSYS will prevent the player from leaving in any direction other than west.

The line

```
             (if (send obj carrying? rusty-key)
```

sends to obj (the actor) a message called carrying? and the parameter rusty-key. We will have defined the method carrying? for the actor object as returning a TRUE when the actor is connected to the object named in the parameter. Thus, this IF-statement will be TRUE if the actor is carrying the rusty key. You can see that ADVSYS, like C, allows well-written programs to be terse yet self-documenting.

The next two lines

```
             (send-super leave obj dir)
             (print "You seem to be missing
                     something!\n"))))
```

are the actions to be taken on the IF statement returning TRUE and FALSE respectively.

The send-super statement sends a message up the class hierarchy to the parent class of the current object. In this case the current object is the storage-room and its super-class is the location class of objects. In the definition of location we will have defined a generalised method for leave, which will disconnect the actor from its current location and connect it with the adjacent one.

Thus the effect of this code is that when the player types W, provided he has taken the rusty key, the game responds `You are in the hallway.' Nine lines of code needing eight paragraphs of explanation is, I think, typical of both ADVSYS and C. It's clear that although ADVSYS offloads the `grunt work' from adventure programming, this doesn't mean you will be able to write the adventure of your dreams in five minutes. Even the fundamental object classes actor and location aren't built-in, but must be defined by the programmer.

You define each class of object, then go on to define particular instances of that class. The instances may have their own properties and methods, in addition to those inherited from the parent class. C programmers, familiar with hierarchical structure definitions, will have no problems with this.

ADVCOM features

Adventure definition files (MYPROG.ADV) are ASCII text files, written using the editor of your choice. They are compiled to data files (MYPROG.DAT), which contain an internal name and version number used by the game's Save and Restore features.

The familiar-looking @include statement instructs the compiler to include another file. The ADVSYS language is highly simplified. The statements available are minimal and the syntax makes no concessions to English. It is LISP-like, with `lots of irritating single parentheses'. Many C programmers will be familiar with editors that have special features to deal with parentheses. For example, the Megamax C editor can find the corresponding opposite of a selected brace or bracket. This feature is extremely useful for ADVSYS.

Operations include integer arithmetic and comparison, Boolean and bitwise logical functions, and random numbers. User-defined functions are supported with arguments, return values, and local variables.

Control constructs include conditional execution, if-then-else, and conditional iteration. Block structures can be defined.

In strings, end-of-line is treated as a space, and multiple spaces are collapsed into one. The special character-pairs \n, \t, and \\ have the meanings expected by the C programmer. For I/O, there is a print statement which provides the plain-vanilla terminal needed by a text adventure. (The line width of the terminal is a compiler option so that it can be set to suit the target machine.) All user input is via the parser, which sets the game variables with the objects to be manipulated and the actions to be performed. The programmer's only other access to what the user actually typed is the yes-or-no expression, which returns TRUE if the player typed a line beginning with Y or y.

Save and Restore are built-in functions. They prompt the user for a filename, and save or load the game's data area.

Restore checks that the file selected by the user corresponds to the adventure name and version being played.

ADVINT features

From the game player's viewpoint, the most obvious features of ADVINT are (of course) those programmed by the game author. ADVINT itself stays in the background. You primarily need to write five handler definitions: INIT, UPDATE, BEFORE, AFTER, and ERROR. The handlers control game activity and form the game's personality. UPDATE, BEFORE, and AFTER are called in sequence with the parser to form the main loop of the game.

The UPDATE handler prepares for the player's next turn, and should describe the player's location if it has changed since the last turn.

The parser is then called. It prompts (:) the player for a command in any of the following formats:

```
[actor,] verb
[actor,] verb dobjects
[actor,] verb dobjects preposition iobject
[actor,] verb iobject dobjects
```

It parses the command and sets the variables $ACTOR, $DOBJECT (the first direct object), $NDOBJECTS (the number of direct objects), $IOBJECT (indirect object) and $ACTION. These variables are used by the handlers. The BEFORE handler is called before the action requested by the command, to inspect the parser variables before proceeding to the action itself.

After the BEFORE handler completes, the action itself is called (or whatever action is stored in the built-in variable $ACTION when the BEFORE handler completes). When the action completes, the AFTER handler is called to handle events that happen only at the end of a successful turn. The ERROR handler is called when the parser detects an error.

Availability

The C source code for the compiler and interpreter, together with the system documentation and a small sample adventure, are available from public domain sources such as CIX (and Chronosoft BBS, see back page for details. ED). Executable programs for various computers are starting to appear too.

The compiler source code amounts to 53K, the interpreter 40K.

However, the total to download is only about 50K (including the sample adventure) because the files are ARCed. In executable form on my machine (an Atari ST) the compiler is 27K and the interpreter 22K. The system documentation is a separate file of 30K.

The C code of ADVSYS appears to have been written for an IBM-PC, with compiler options (#ifdef MAC) for a Macintosh. I was intrigued as to whether I could compile the programs on my Atari ST using Megamax C.

In fact, ADVCOM compiled and linked successfully with no modifications at all. For ADVINT it was necessary to change a number of bdos(x) calls in ADVTRM.C which perform character I/O. I substituted various Atari BIOS calls until the terminal behaved properly. However, I discovered from info posted on CIX that it's possible to directly substitute Atari gemdos(x) for IBM bdos(x) calls.

From CIX I also obtained info on several bug fixes, and much other useful stuff that originated on BIX in the US.

With six or seven source files to compile and link I found the Megamax Make function most useful and time-saving. I was extremely pleased to find that I could compile and play the sample adventure on my ST.

There is a conference on CIX dedicated to ADVSYS, for users to swap experience, hints and tips. The download area contains the ADVSYS source code and several executable programs. No doubt in the near future some additional public-domain game source code will appear here.

The possibility exists for more than one programmer to work simultaneously on different parts of a large adventure, and there is word of a possible UK-vs-US challenge to produce the best ADVSYS-based adventure! If there is sufficient interest I shall be pleased to contribute further ADVSYS reports to "C Vu".

Details of how to contact Martyn can be found on the back page, with your questions, suggestions, etc. We look forward to hearing more about ADVSYS in future issues -- ED.

Structure, Part 1

By Colin Masterton

Introduction.

In this series of articles I'll talk a bit about structuring 'C' programs. Although aimed at the fairly recent convert to 'structured' languages, the series may contain points of interest to more experienced users. In this first part we'll try to identify what structured programs are.

The Holy Grail.

Like some Holy Grail, we talk about this thing called structured programming. Everyone has heard about it, few can explain it, fewer still seem able to produce structured programs. Our students are told about the benefits of using structured languages like Pascal, but few seem completely at home with the whole idea. It's not difficult! Most of it is common sense really. Like everything - you get better with practice. Let's have a look at this magic and try to dispel some of the myth.

Engage brain.....

One good starting point which is definitely worth mentioning again - leave the keyboard alone! I'm sure that the big problem with many new programmers (and I was no exception) is their eagerness to get coding. This is not a good plan. When you're typing at the keyboard - coding or whatever - you are too close to the problem to see the 'big picture'. It's too late. Within the first few minutes all hope of structure will have gone. So STOP. Think a little before you code.

Now let's see if we can tell how to identify structured programs:- that should make the task of achieving them that much easier.

What's in a language ?

Surely, if we use a language like Pascal or C then our worries are over ? They are designed as structured languages. They force certain logical constraints on us. Why need we say more ?

Let's start by asking some questions.

. What is structure ?

- Why use structure ?
- How do we identify good structure ?
- How do we produce good structure ?
- Can we define a set of rules to help us ?
- What makes a good procedure ?

We'll have a look at each of these questions and, as we build an understanding about what is and is not structure you will begin to see how to achieve it.

What is structure ?

Perhaps it's easier to start by saying what is NOT structure. Let's take a popular example - that of making a cup of tea. A simple enough job and one made up from a set of clearly definable steps. Here are some of the things we would have to do:-

11	Pour tea in cup.	9	Put tea in pot.
4	Fill kettle.	6	Warm tea pot.
7	Put water in teapot.	12	Stir cup.
8	Open tea caddy.	2	Check if kettle filled.
10	Put milk in cup.	1	Check if teapot empty.
3	Turn on tap.	13	Drink tea.
5	Boil kettle.	14	Pick up kettle.

Well, most of the things that need doing are there, but there's not much structure to it I'm sure you'll agree. It doesn't make sense to warm the pot AFTER we've put the tea in it! Clearly, the actions are in the wrong order. There's another problem too - we can't get any sort of overview of what's going on - we don't see the general outline of the task; we've got to look at lots of detail to see what's going on.

Now here's the same problem presented in a different way:-

```
Pick up teapot, open lid, check if full, close lid.
Pick up teapot, open lid, empty old tea, close lid.
Pick up kettle, open lid, check if full, close lid.
Pick up kettle, open lid, place under tap, turn on tap.
Check if kettle full, turn off tap, close lid.
Plug in kettle, switch on power,
     wait for kettle to boil.
Switch off switch, pick up kettle, open lid of teapot.
Pour in water, warm pot.........
```

We've been going for ages and we're no-where near a cup of tea yet! What's wrong this time ? The general actions are correct and in the correct order but the purpose is obscured by detail. Open and close lids everywhere! If you didn't know what was going on you'd be asking 'What's going on here ? What is he trying to do ?'

Notice that in both these cases the actions are correct. The result will probably be a cup of tea. Structure therefore, is nothing to do with whether or not the outcome is correct. Witness the hundreds of totally unstructured programs there are around. You could write an infinite number of programs to achieve a working result and none of them need be structured.

However, structure does assist in reaching a functioning program and also in modifying a program.

Why do we use structure ?

Realise that no program will work perfectly first time. Therefore, take steps to make life easy on yourself. Modifying a program is something done during development too. It's not just something that happens years later. If you think you'll remember what's going on after a couple of months then you're probably wrong. Conditions change. The problem you started off writing a solution to may not end up as the same problem. Either you will see better ways to do things as you get further into the project, or the person you are writing it for will change his mind. It makes sense to build redundancy and versatility into your programs.

Firstly for your own use, so that you can easily reuse the things you have written for another project. Secondly, as I have said, the program you write to make one cup of tea is likely not to be for just one cup of tea at the end of the day, eg. If we decide to use a kettle which doesn't have a lid, think of the number of places we need to change a line in our second list of actions.

If we want to boil two kettles at once or make ten cups of tea - how can we handle that? (Especially if our teapot can only make five cups of tea at once.)

Special case ?

Try to take a step back from the problem, avoid the detail and look at the broad steps. Try to think ahead to likely future changes and try to identify the 'special cases' implicit in what you are doing, eg. making a cup of tea is a 'special case' of making 'n' cups of tea where:- n = 1, and boiling a kettle is a just boiling 'n' kettles where:- n = 1.

This last point is true of almost every problem you will face. Sometimes situations are such that you know you need never consider more general cases. Often, however, realising that something is a special case is a positive help. Let's look at another example:- find the average of 10 numbers between 1 and 100.

This is a special case of finding the average of 'n' numbers between 'min' and 'max' where:- n = 1, min = 1, and max = 100

If you recognise these points early on in your thinking, it will alter the way you consider each step. Boiling the kettle is no longer just something that needs to be done at some point. It is a self contained task which we will request simply as 'boil kettle'. We need not concern ourselves with lidless kettles or about whether it was part full or empty. As a first step, all we need know is that there is a task (procedure, function or subroutine) which, if we call it, will result in us having a boiled kettle.

Of course, at some point we must put in all the necessary detail into boil_kettle(), but let's not think about that just now.

If you can achieve just these two objectives:-

- Concentrate on high level actions ignoring the detail.
- Recognise special cases.

then you have made a good start in structuring.

In the next part we'll begin to formalise our definition of structure and try to produce a set of maxims to work with.

/****************************/

TITLE	:	PROFICIENT C
AUTHOR	:	Hansen, Augie
PUBLISHER	:	Microsoft Press.
ISBN	:	1-55615-007-5
PRICE	:	19.95

A grand sounding title, and a book which, according to the sleeve notes, is intended for advanced or intermediate programmers, yet one which contains much useful information for anyone programming on an IBM type machine under MSDOS.

Although C does not require MSDOS nor an IBM type machine, it is inevitable I suppose, that a book emanating from the Microsoft stable will be heavily slanted in that direction. This one is no different but that does not detract from the usefulness of the text.

The author is clearly used to working under UNIX and makes frequent reference to the points which would require attention in porting software to this environment. The applications however are undoubtedly DOS applications and, towards the end of the book, involve such close contact with the IBM hardware that perhaps a more honest title would have been 'Proficient C on an IBM PC'.

That aside, this is a well constructed text which covers many aspects of building fully functioning programs in C. Whilst the latest Microsoft C compiler is used by the author, there is little to compel users to follow his lead.

Little time is wasted in explaining, justifying or teaching C. The first chapters cover the DOS programming environment and the connection between DOS and C. It should be pointed out that at least some of the comments made in this respect refer to the Microsoft DOS to C connection and other compilers may differ in some areas.

The style of the book is pleasant and easy to follow. A useful list of all functions presented is included as an appendix. Functions are clearly defined, well commented and documented. Most of the functions presented are used as building blocks later in the book, eventually leading to a system of screen windows which can be fully controlled by the user.

As always, one of the best ways to improve your own C style is to look at the way others program. This book presents no slick-and-tricky super C constructs but does demonstrate well structured code and good use of library routines. It is reassuring too, to see methods employed for making BIOS calls and obtaining information from DOS such as program name. Reassuring, since, if Microsoft adopt such approaches, we can be sure that it is safe for the rest of us to do so too.

Text books never seem cheap, but, in terms of value for money, I would suggest that this is indeed a useful book to add to a reference shelf.

TITLE	:	DEBUGGING C
AUTHOR	:	Ward, Robert
PUBLISHER	:	Que Corporation.
ISBN	:	0-88022-261-1
PRICE	:	19.95

"Why a book on debugging ?" the author asks in the first chapter. Good question ? Well anyone who had attempted 'Hello, world' and anything more advanced in C would probably be able to give you a good enough answer!

Of course there IS a need for a book about how to fault find software and, although this one is not the whole story, it serves as a good starting point.

The author is quite clear about several points; lets not be ashamed of having to debug programs, rather, come out of the closet and do it properly. Lets carefully categorize the types of problem we're chasing and arm ourselves with the right tools for the job.

Strangely, he seems to suggest that its not a good idea to spend time thinking about the problem while staring at a listing or a screen, but recommends getting in and 'instrumenting' the program to gather more information. (On this last point I feel I may have to beg to differ slightly.)

Although some of the syntax is a little puzzling at times, the author clearly knows what he's talking about. He has been there. Wrongly dimensioned arrays and errant pointers are not new to him. He clearly explains the possible outcomes of such, and other types of C problem and then procedes to present ways to track them down.

His approach is one of accepting that a problem is there and then finding it, rather than preaching about the way you should have written the thing in the first place to avoid the problem.

The author presents several fragments, and a fully operational on-line debugging system. Not a full symbolic debugger but extremely useful nonetheless.

The author takes time to explain just why bugs in C can be so tiresome to locate and is quite thorough in this explanation. In fact, anyone stopping after the first couple of chapters would be quite justified in assuming that the task was in any case impossible, such is the horror and variety of the problems Ward presents.

The text covers technique of stabilizing, locating, tracing and fixing bugs.

He also discusses the use of source level debuggers and compiler/interpreter environments.

In his final chapter, Ward presents a challenge to compiler writers which they would all do well to read. In line with his philosophy of accepting that bugs will appear, he suggests that more in the way of debug facilities could and should be available from compilers.

A worthwhile read and handy to have on the shelf. No one will be an expert in debugging C after reading this book but at least Ward provides the confidence that the bugs can be found - somehow!

/****************************/

THE ITALIAN CONNECTION!

The following letter has been received by Martin Houston:-

Padova 10 Set 1987

Dear friend,

We are a group of 'C' language enthusiasts, and have just created an official club in Italy. We already have the support from the ICUG of Mc Pherson, Ks and of i2u (the Italian Unix User's group).

Our main aim is to contact and communicate with people like you, for sharing informations ideas and also some public domain software e/o compiler libraries written in C.

Our associates vary from students to computer professionals, but the association isn't lucrative. The principal activities are the maintaining of our software library, the publication of a quarterly newsletter (but we plan to make it bimonthly), the exchange of information with Italian and American universities and colleges.

Today the group owns an MS-DOS machine (Olivetti M-24), and a AT&T 3B2 mini running Unix Sys V, and a Epson PC AX with SCO Xenix Sys V. We have a common software library for the three systems, and we plan to expand them, also with your help.

The annual subscription is 20,000 Lire (= 10 pounds) for

Italian subscribers, 40,000 (= 20 pounds) for foreign subscribers. This fee covers the reproduction and P&P of the newsletter, and the access to the software library.

Anyway, we are interested in you contacting us, giving us suggestions, etc. Our newsletter is open for contributions from you, and we'd like to hear what happens in your country.

I hope to hear from you, and thank you for this opportunity.

Micro C Users' Group
c/o Massimo Cesaro
Via Monte Vodice n. 4
35138 Padova
Italy

If anybody is interested in liasing with the Italians, then drop Martin Houston a line, or write direct.

/****************************/

Adventures in C
A few thoughts on using C to write Adventure Games
By Martin Houston.

In this article I hope to reveal some of the structuring features of C that would enable an adventure game to be written in an easy to understand and easy to modify form. To my mind one of the most useful features of C in writing and adventure type program is the ability to store and later use pointers to functions just like any other data. This allows a very elegant and generalised approach to be made to the problem of processing the phrases that the player types into the adventure so that they have an effect on the adventure 'world' represented inside the computer.

The adventure parser 'adshell' presented here should serve as a backbone for an adventure program: all that is missing is an adventure to work on. The adventure writer will have to devise some way of representing the world of the adventure and providing action functions to affect this world in response to the recognition of keywords.

The writer of the adventure has simply to supply a set of action functions written in C that interact with each other in a defined way through the mechanism described below and by the modification of global variables to control each others action.

In the command 'HIT DRAGON' the action of the 'HIT' handling function obviously depends on the thing being hit and the action of the 'DRAGON' function must have fore-knowlege that the dragon is being hit. Each must leave some indication of how the two will interact to cause a change in the state of the adventure as defined by the global data structures.

The parse function will do no syntactic analysis itself. It simply seperates each word of the input string and looks it up in the action table to find a function to call. If no match is found for a particular word the parser just goes on to the next. Some words in the input by be considered as parameters to the action functions. In the above example of 'HIT'and 'DRAGON' it would make sense to have a handler for hit that assumes that the next word is the thing to be hit and acts accordingly.

The parser is invoked from the main() of the adventure program as

```
parse(line_of_input, action_table)
```

If parse returns false (0) then NONE of the input line could be understood and the main program can print an appropriate message. A return value of non zero will indicate that the input has affected the adventure world in some way.

The action functions are stored in a table of strings and function pointers. The function pointer is called if the string token matches the current word. Matches are case sensitive in the code as provided but this can be changed by adding a 'tolower()' call in the building of the array of words on a line which forces all user input to lower case before comparison is carried out.

Each function will be called with an integer and a string parameter: the first parameter will be a pointer to the complete string that is being parsed. The second parameter is an integer that defines the offset of the whole line string that can be used to access the word that caused the function to be invoked.

```
func(line_of_input, index_of_token)
```

The return value of func is very important to parse. It must be either a positive or negative integer. Parse will make the next word it considers the index of the token just processed + the return value of the processing function. This means that a negative value can cause the parser to back up and a positive value other than 1 can cause words to be missed. A return value of 0 will cause the function to be re_invoked - useful if the function wishes to re-enter itself from the top.

A dummy application for parse is provided that recognises the words 'help' and 'exit' only but shows the principles of use.

I do not have the time or the imagination to develop a full blown adventure but I hope that routine will prove a useful tool for those that have.

```
/*
 * Adventure parser shell header file.
 * By Martin Houston for CUG(UK)
 * 25/5/86
 */

/* default sizes for various things */
#define WORDLEN 25     /* no words longer than 25 chars */
#define LINELEN 255     /*  < 255 characters in one input from
user */

/* declaration for function look up table structure */
struct look_up_func
{
     char token[WORDLEN];
     int (*action)();     /* a POINTER to a function */
};

/*
 * Adventure parser shell code file.
 * By Martin Houston for CUG(UK)
 * 25/5/86
 */

#include <ctype.h>
#include "adshell.h"

#define SAME(A,B) (!strcmp(A,B))
#define TRUE 1
#define FALSE !(TRUE)
#define NULL  (char *)0


int parse(line, tab)
char *line;
struct look_up_func tab[];     /* array of structs */
{
     int acted = FALSE;     /* we have not done anything yet */

     static int wordstarts[LINELEN/2];
     /*
      * As each word is seperated from the next by
      * whitespace there can only be at most LINELEN / 2
      */

     static char words[LINELEN/2][WORDLEN];

     int slot;     /* slot in table */
     int lindex;     /* maximum used slot in table */
     int cn;          /* character counter */
     int x;          /* dogsbody variable ! */
     char *str;     /* input string */
     char c;          /* next character */

     int fpslot;
     struct look_up_func *fp;

     char between_words;

     /*
      * Build up tables of words and word starts
      */
     str = line;
     between_words = TRUE;
     for(lindex=0, slot=0, cn=0; lindex<LINELEN; lindex++)
     {

          if(((c = *str++) == '\0') || isspace(c))
          {
               if(cn > 0)
                         {
                    /* terminated a word */
                    strncpy(words[slot],
                         &line[wordstarts[slot]], cn);
                    words[slot][cn] = '\0';
                    cn = 0;
                    slot++;
               }
               between_words = TRUE;
               if(c == '\0')
               {
                    break; /* no more input line */
               }

          }
          else
          {
               if(between_words)
               {
                    /* started a word */
                    wordstarts[slot] = lindex;
                    between_words = FALSE;
               }
               if(cn < WORDLEN)
                    cn++; /* count a character */
          }
     }


     /* now try to find a match */
     for(x=0; x < slot; )
     {
          fpslot = 0;

          while((fp = &tab[fpslot++]) != NULL)
          {
               if(SAME(words[x], fp->token))
               {
                    acted = TRUE;
                    x += (*fp->action)(line,wordstarts[x]);
                    if(x < 0)
                         x = 0;
                    break;
               }
               if(SAME(fp->token, ""))
                    break;
          }
          /* if no match was found above ignore this word */
          if(SAME(fp->token, ""))
               x++;
     }
     return acted;
}


/*
 * Adventure parser shell dummy use file.
 * By Martin Houston for CUG(UK)
 * 25/5/86
 */
#include "adshell.h"

/*
 * Must declare all funcs we will put in table explicitly.
 */
int help();
int fexit();

struct look_up_func tab[] =
{
     "help", help,
     "exit", fexit,
     "", (int (*)())0
};
```

```
int help(line, index)
char *line;
int index;
{
     printf("No help for the wicked!\n");
     /* tell parser to examine next word */
     return 1;
}

int fexit(line, index)
char *line;
int index;
{
     /* allows get out from prog */
     exit(0);
}

main()
{
     char line[LINELEN];

     while(1)
     {
          printf("What next ? ");
          gets(line);
          if(!parse(line, tab))
          {
               printf("I dont understand you !\n");
          }
     }
}
```

CALLING ALL 1986 CUG MEMBERS

Hello there! Welcome to the newly re-formed CUG(UK).

The original C Users' Group run by Mr Leon Heller to which you previously belonged stopped doing anything back in March 1986. I myself belonged to the group as an ordinary member. Unfortunately Mr Heller became too busy to administer the original CUG so the group collapsed through lack of communication.

In June of this year Personal Computer World magazine finally got around to publishing and article about CUG that I had sent them over a year previously! As my address was given as the contact for the group I was suddenly snowed under with enquiries about CUG without having a group to enrol them in. I therefore decided to re-form and run the C Users' Group. The aims of the group are outlined elsewhere in this magazine.

As I have come into possesion of the 1986 membership list I have decided to send all of the 1986 members this first copy of the new group newsletter as a goodwill gesture. If you wish to become a member of the re-formed CUG then please send your 10 pound subscription payable to "CUG(UK)" to:

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

THE CUG(UK) ONLINE SYSTEMS

Birmingham

By Martin Houston

CUG has come to an arrangement with the Chronosoft Bulletin Board to allow the keeping of the C source code library and magazine articles 'online' so that anybody with a suitable modem & comms software can donate to and withdraw from the source code library and make C related enquiries without the delay and bother of going through the post.

The use of the board to CUG members is at the moment free. Colin Earl the system operator knows the names & addresses of all paid up members and you will be accorded CUG member privilidges accordingly.

The board, which runs the WILDCAT! BBS system is available on

021 744 5561 for V21/V23 8 data bits No parity 1 stop bit.

It is menu driven with online help so if you ever get stuck just type a '?' at the prompt. To get you started there is an introductory piece about CUG in the main bulletin menu. I am a regular user of the board so just mail me if you have any queries.

One final note: the board is quite popular and my be difficult to get onto when the phones are cheap rate. If group membership grows enough then we may be rich enough to have a machine all of our own. The more of you out there that use the service the more likeley this will be.

Guildford

By Phil J Stubbington

In addition to the Chronosoft board, there is also a section on CIX (Compulink Information Exchange) devoted to CUG(UK). Unfortunately, CIX will cost you money (in addition to your 'phone bill) - for details of costs, and other services that they offer please contact:-

Compulink,
67 Woodbridge Road,
Guildford,
Surrey,

When you join CIX you will receive all the relevant details, but just for reference the software used is called CoSy, and runs 24 hours a day, seven days a week on a Unix system (so expect to get dropped frequently!) on:-

0483-573 338 & 0483-573 337,
8 data bits, 1 stop bit, no parity

I frequently use CIX (nickname=philip) so you can easily contact me. It is envisaged that the 'c_users_group' conference will offer the same facilities as the Chronosoft board, but please be patient!

The Back Page

Well, here we are. The back page! I hope you have found the magazine interesting, and will tell your friends all about us. If you have a particular area of interest, why not write an article about it? How about a review of your favourite editor, compiler, or debugging tool?

Now, for the addresses. For all general enquiries about CUG(UK) including membership details, and those Unix and Xenix problems, please contact:-

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

For problems with Atari ST's, MS-DOS, OS-9:-

Steven W Palmer,
L. J. Electronics Ltd,
Francis Way,
Bowthorpe Industrial Estate,
Norwich,
NORFOLK,
NR5 9JA.

For enquiries about the magazine, either contact me via Martin Houston, or mail me on CIX (nickname=philip).

To contact Martyn Dryden, the author of our article on AdvSys:-

Telecom Gold: 74:ELP205
CIX: mdryden