# Lattice C 5

*the C Compiler for your Atari ST Computer*

## Volume I

*User Manual*

**Requires:**

- ✓ Atari 520ST upwards
  *(1M+ memory advised)*
- ✓ Disk drive
  *(2 floppies or hard disk advised)*
- ✓ Mouse

HiSoft
High Quality Software

# Lattice C

The C system for your Atari ST

## Volume I
## User Manual

**Copyright © HiSoft & Lattice, Inc. 1990, 91**
**Published by HiSoft**

**Version 5**
**First edition March 1990** (ISBN 0 948517 29 8)
**Second edition April 1991**

ISBN for this volume 0 948517 37 9

ISBN for complete 3 volume set 0 948517 28 X

Set using an Apple Macintosh™ and Laserwriter™ with Microsoft Word™ and SuperPaint™.

# Table of Contents

# Introduction

Welcome to Lattice C Version 5, one of the most powerful and flexible programming environments available for the Atari ST/TT range of computers. This new version of Lattice C is based on proven Amiga and PC compilers, coupled with a complete suite of programming tools and libraries, resulting in a C development system that is unparalleled in the speed and the quality of its object code, while being a joy to use for the beginner and the professional alike.

The documentation for Lattice C 5 is divided, like Gaul, into three parts: this volume gives an overview of the whole system and details the working of the compiler, linker, assembler and all the associated tools while the other two volumes document the many library functions available, both generic (Unix, ANSI and Lattice) and Atari ST specific (GEM, AES, VDI etc.). There is a wealth of information here and you should not expect to assimilate it all in one reading - you are encouraged to work through the rest of this chapter and then to use Lattice C 5 in anger, dipping into the various manuals for reference, as and when the occasion arises.

Before we proceed to using Lattice C 5 for the first time there are a couple of important things that you must know about ...

## Making a Backup

You have several ways of making a backup of the Lattice C 5 disks. You can use the disk copying function of the Desktop to duplicate the disk, or you can use one of the many disk copiers available.

However you do it, *please make a backup* and then store the master disks in a safe place, away from moisture, extreme heat or cold, magnetic fields (televisions, telephones etc. give off radiation harmful to disks), strong light, coffee and, above all, children and dogs! If you damage your master disks we will charge you a handling fee for re-copying them.

## Technical Support

We are striving continually to make this product better, so we are very receptive to suggestions about how Lattice C 5 could be made more useful to you. If enough people ask for the same types of features, the likelihood is high that such features will be implemented in a future release of the package.

To take advantage of the support we offer, even if only to receive details of the newest major revision of the program, you must have sent us your registration card. You will then be sent any new information about Lattice C.

You must also quote your serial number for technical support, you may find it useful to make a note of it here:

| Serial No. |
|---|

See **Appendix J** for more details of Technical Support. Please ensure that you have read this section *carefully* before contacting us for technical support as it also describes some of the common problems and how to solve them.

## What is Lattice C 5?

So that you can get the most out of these manuals and the software, here is an overview of the complete Lattice C 5 system.

Lattice C 5 is a C development system comprising a host of tools and utilities allowing you to create, compile, link and run programs on the Atari ST. It conforms closely to the new ANSI standard for the C language and includes many extensions that give you a great deal of flexibility in the C environment. The package also contains the most extensive set of library functions (both generic and ST-specific) available on the Atari ST computer.

### Editing C Source Code

You can create and edit C programs using any editor of your choice, as long as the editor can save the source code as plain ASCII characters. We provide, and recommend, an editor, called EdC, which is GEM-based, easy-to-use and is a complete, visually-oriented shell.

We also supply a derivative of EdC, called LC.PRG, which integrates the editor and the first phase of the Lattice C 5 compiler, giving a fast edit-compile-edit cycle. This is usable on machines with at least 1Mb of memory.

### Compiling and Linking C Source Code

In order to turn your C source code into an executable program, you need to compile and link it. If you are not using the integrated system (LC.PRG), you will need to use the compiler driver (LC.TTP) either from the Desktop or from a command line shell (CLI) - we supply a MS-DOS style shell which is known as Batcher.

The Lattice C 5 compiler operates in two phases, the first phase (LC1) performs pre-processing and parsing of your C code into an intermediate format known as a quad (.Q) file. The second phase (LC2) takes this quad file and produces 680x0 object code in a file (.O extension), ready for linking.

The linker, CLink, creates a runnable, machine code program by combining one or more object files produced by LC2 with the relevant supplied libraries.

The integrated system lets you, with a single key-press, invoke the first phase of the compiler (which is built in), LC2 and then the linker, producing an executable program, ready to run from the editor.

From a shell, the LC driver (LC.TTP) provides one command line to control and invoke LC1, LC2 and the linker.

### Debugging the Program

We supply a low-level debugger (MONST2C.PRG) that will help the more experienced of you to investigate the operation (or not!) of your program. The debugger has some higher-level features, normally only found in source level debuggers, allowing access to your source code on a line number basis. We hope to produce a full source level debugger at a later stage.

### Improving the Program

We supply a number of powerful tools allowing you to create good-looking, fast and compact programs with a minimum of effort.

WERCS is a resource editor giving full access to GEM, making it easy to incorporate menus, dialog boxes and icons. It is compatible with earlier resource editors and also lets you include graphics from art packages.

The assembler, ASM, is a full 680x0 macro assembler, tailored to the Lattice C 5 environment, letting you exploit the 680x0 family of processors to the full.

To achieve maximum performance without resorting to assembly language, Lattice C 5 incorporates a global optimiser, GO. GO gives you the option of increasing the performance of your program with no programming effort, although it can take some time to complete its task!

### Bells and Whistles

Lattice C 5 also comes with a plethora of minor tools, whose usefulness will depend on your own needs and experience. These include such things as a reset-proof RAM disk, a librarian, an object module disassembler etc. which are all fully documented in **The Lattice C 5 Tools** chapter.

# A Lattice C 5 Tutorial

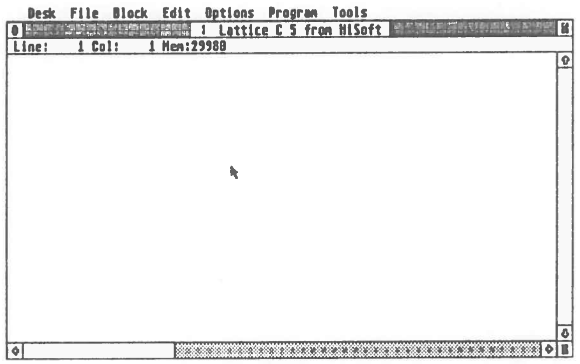
## Lesson 1 - Your First Program

We are now going to guide you through your first experience with Lattice C 5 (at least, we hope it's your first experience - you haven't been too impatient, have you?). First of all, you must install Lattice C 5 for your system. The method of doing this depends on your system (how much memory you have and whether you have a hard disk or not) - we have documented the installation process in a separate document (the **Installation Guide to Lattice C 5**) since there are great number of different possible configurations and the installation may change as we upgrade the package.

Lattice C 5 will run on 512K machines but you would be well advised (as you will soon see) to upgrade your machine to at least 1Mb of memory so that you can take full advantage of the package's features. We will take you through a simple program on both 512K and larger computers to show you the different working environments.

### 520ST/STE Owners

First of all, find the EDC.PRG program (wherever you have put it on installation) and, from the GEM desktop, double-click on the EDC.PRG icon to run the program. The following screen will appear:

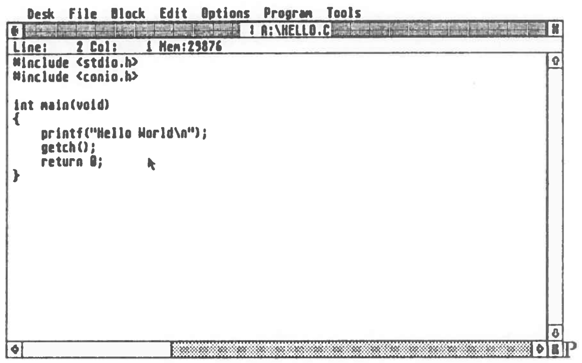

Desk File Block Edit Options Program Tools
1 Lattice C 5 from HiSoft
Line: 1 Col: 1 Mem:29988

*The EDC Editor Opening Screen*

Now type the following program using the keyboard in the normal way and pressing Return at the end of each line:

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
  printf("Hello World\n");
  getch();
  return 0;
}
```

Your screen should now look like this:



```
#include <stdio.h>
#include <conio.h>

int main(void)
{
    printf("Hello World\n");
    getch();
    return 0;
}
```

*The Hello World Program*

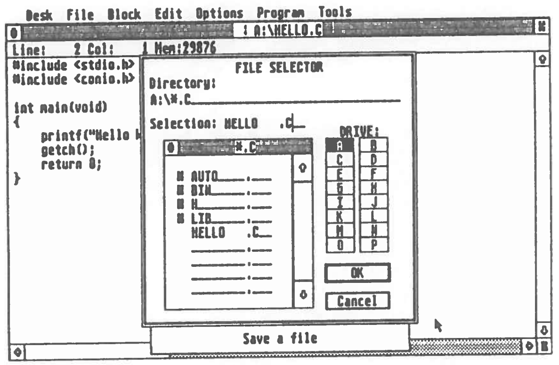
Before we compile this complicated program we must save it to disk because, on a 512K machine, we have to quit the editor to compile/link etc.

The best place to save your source files is the Lattice C 5 working disk or your hard disk if you have one. So press Alt-S or click on Save As... from the File menu and the file selector will appear.

The appearance of the file selector will depend on which one you have installed in your system. If your operating system is TOS 1.4 or greater, you have, automatically, an extended file selector with drive buttons such as that shown in the screen shot on the next page.

However, if you have an earlier version of the system ROMs, you will see, unless you have installed a new one, a much more primitive file selector, without drive buttons. You can replace this system file selector, if you wish, with the extended HiSoft File Selector (HFSEL) by placing HFSEL in your AUTO folder - see the **Installation Guide to Lattice C 5** for details.

When the file selector appears, type in HELLO.C and click on the relevant drive button so that the box looks like this:



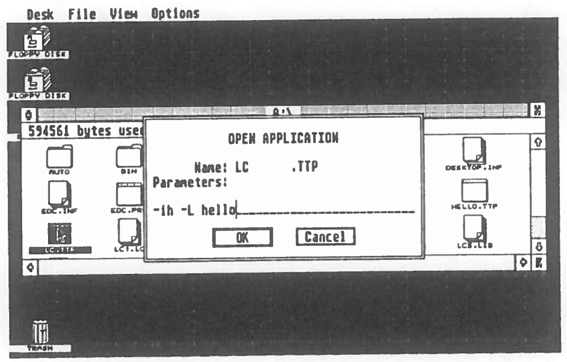
```
#include <stdio.h>
#include <conio.h>

int main(void)
{
    printf("Hello
    getch();
    return 0;
}
```

*The File Selector*

Quit the editor using Alt-Q or by selecting Quit from the File menu - you should now be back on the GEM desktop. Double-click on LC.TTP to run the compiler driver - a box will appear, type in:

```
-ih -L hello
```

so that the box looks like:



*The LC.TTP Command Line for HELLO.C*

The -ih means 'find the include files in a directory called h' and the -L means 'link after compilation'.

Now press Return to start the compilation and link. Assuming this proceeds correctly you should see the following messages on the screen:

```
Lattice Atari C Compiler Copyright © 1990 HiSoft & Lattice, Inc.
All rights reserved - Version 5.04.00

Compiling hello.c
Module size P=0000001A D=0000000E U=00000000

Total files: 1, Compiled OK: 1
Linking hello
Clink Copyright © 1990 HiSoft & Lattice, Inc.
All Rights Reserved - Version 1.11


CLINK Complete - Maximum code size = 6464 ($00001940) bytes
```

*The Compilation and Link of HELLO.C*

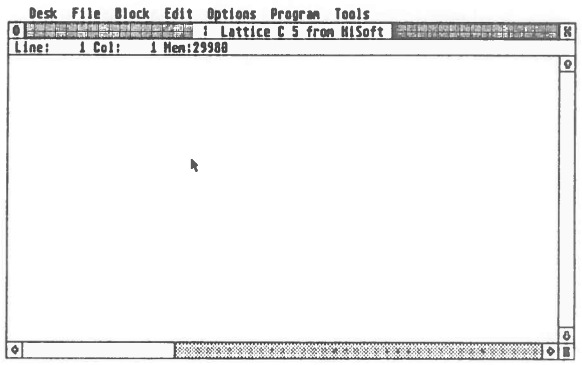
If you do not achieve a successful compile & link, check that you have installed the system correctly and that all the various tools and libraries are where we recommend.

After this you will have a file called HELLO.TTP on your working disk - double-click on this, type Return when the box appears and you should see Hello World appear at the top of the screen, hit a key and the GEM desktop will re-appear.

Congratulations - you have created, compiled, linked and executed your first Lattice C 5 program!

## 1040ST/STE, Mega, TT Owners

First of all, find the LC.PRG program (wherever you have put it on installation) and, from the GEM desktop, double-click on the LC.PRG icon to run the program. The following screen will appear:

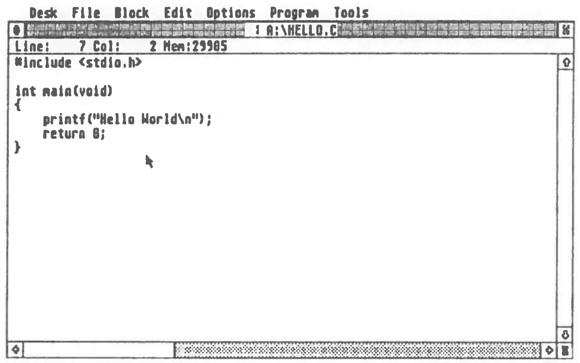

Desk File Block Edit Options Program Tools
1 Lattice C 5 from HiSoft
Line: 1 Col: 1 Mem:29988

*The LC Editor Opening Screen*

Now type the following program using the keyboard in the normal way and pressing Return at the end of each line:

```
#include <stdio.h>

int main(void)
{
  printf("Hello World\n");
  return 0;
}
```

Your screen should now look like this:



```
#include <stdio.h>

int main(void)
{
    printf("Hello World\n");
    return 0;
}
```

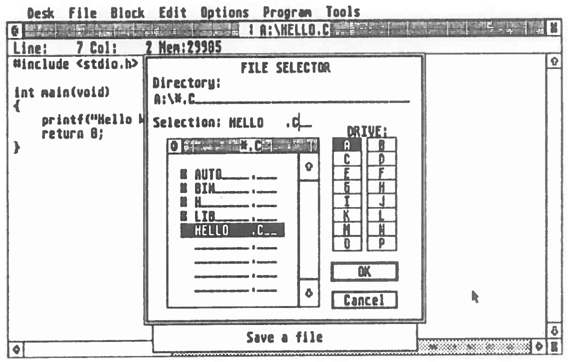
*The Hello World Program*

Before we compile this complicated program we really ought to save it to disk, in case anything goes wrong The best place to save your source files is the Lattice C 5 working disk or your hard disk if you have one. So press Alt-S or click on Save As... from the File menu and the file selector will appear.

The appearance of the file selector will depend on which one you have installed in your system. If your operating system is TOS 1.4 or greater, you have, automatically, an extended file selector with drive buttons such as that shown in the screen shot on the next page.

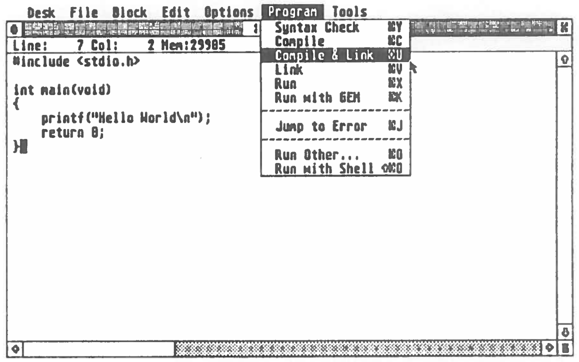
However, if you have an earlier version of the system ROMs, you will see, unless you have installed a new one, a much more primitive file selector, without drive buttons. You can replace this system file selector, if you wish, with the extended HiSoft File Selector (HFSEL) by placing HFSEL in your AUTO folder - see the **Installation Guide to Lattice C 5** for details.

When the file selector appears, type in HELLO.C and click on the relevant drive button so that the box looks like this:



*The File Selector*

To compile and link your program type Alt-U or select Compile & Link from the Program menu as shown below:



*The Program menu before Compile & Link*

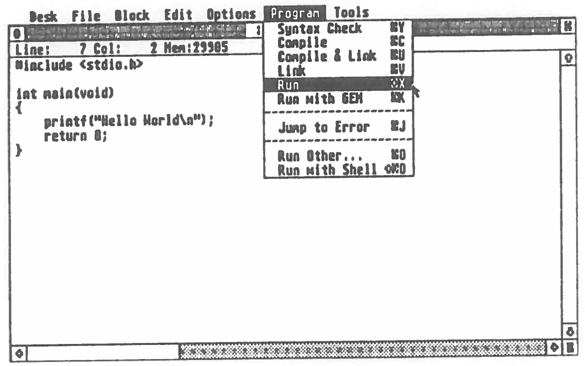
The compilation and link will now proceed immediately and, assuming it is successful, you will then be asked to press a key to return to the editor. If anything goes wrong, check that you have typed in the program correctly and that you have copied the EDCTOOLS.INF file as described in the **Installation Guide to Lattice C 5** - otherwise the environment may not be set up.

A correct compilation and link will generate messages something like this:

```
Lattice Atari C Compiler Copyright © 1990 HiSoft & Lattice, Inc.
All rights reserved - Version 5.04.00

Module size P=0000001A D=0000000E U=00000000

Total files: 1, Compiled OK: 1
Linking hello
Clink Copyright © 1990 HiSoft & Lattice, Inc.
All Rights Reserved - Version 1.11


CLINK Complete - Maximum code size = 6464 ($00001940) bytes

Final output file size = 6446 ($0000192e) bytes
```

*The Compilation and Link of HELLO.C*

After this you will have a file called HELLO.TTP on your working disk - to run the program first ensure that Run with GEM on the Program menu is *not* selected and then type Alt-X or click on Run on the Program menu, then type Return when the box appears and you should see Hello World appear at the top of the screen, hit a key and you will be returned to the editor.



```
Line:    7 Col:    2 Mem:29905
#include <stdio.h>

int main(void)
{
    printf("Hello World\n");
    return 0;
}
```

*Running your First Program*

Congratulations - you have created, compiled, linked and executed your first Lattice C 5 program!

## Summary of Lesson 1

So, what have we learned so far?

- 520ST/STE users cannot use a fully integrated system because there is not enough memory to support this - you must create/edit your program using EDC.PRG, exit and compile & link using LC.TTP, typing in the necessary command line.

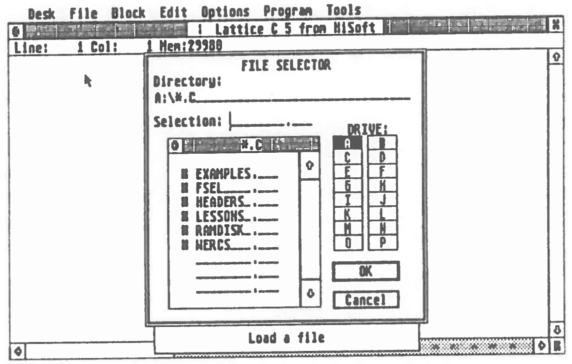
- *Important note for TOS 1.0 users:* this early version of the ST ROMs upper cases the command line typed in to .TTP programs. This can cause problems since the LC.TTP command line is *case sensitive* (because there are so many options). To get around this we have added an extra option, -?, that prompts for a further command line to be input, thus bypassing the ROM command line and allowing mixed upper and lower case to be used. Use -? if you have TOS 1.0 and need to use command line options that are case sensitive.

- 1040ST/STE and Mega users can take advantage of the integrated LC.PRG, creating, editing, compiling, linking and running their programs all from within the editor.

## Lesson 2 - We all make mistakes

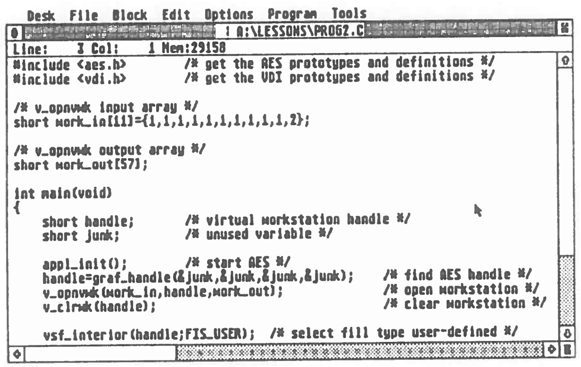
In this lesson we are not going to differentiate between 512K and 1Mb+ users and we shall simply tell you to compile, link and run your program - please refer to Lesson 1 if you are unsure how to do this.

Run EDC.PRG (512K users) or LC.PRG (1Mb+ users), insert your working disk and type Alt-L or select Load from the File menu



*The Load File Command*

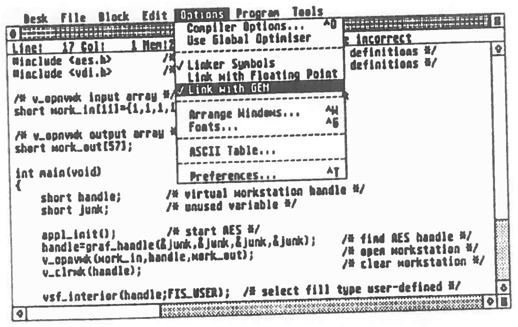
Single-click on the Lessons directory to open it and then double-click on PROG2.C to load the program into the editor.



```
#include <aes.h>          /* get the AES prototypes and definitions */
#include <vdi.h>          /* get the VDI prototypes and definitions */

/* v_opnvwk input array */
short work_in[11]={1,1,1,1,1,1,1,1,1,1,2};

/* v_opnvwk output array */
short work_out[57];

int main(void)
{
    short handle;         /* virtual workstation handle */
    short junk;           /* unused variable */

    appl_init();          /* start AES */
    handle=graf_handle(&junk,&junk,&junk,&junk);     /* find AES handle */
    v_opnvwk(work_in,handle,work_out);               /* open workstation */
    v_clrwk(handle);                                 /* clear workstation */

    vsf_interior(handle;FIS_USER);  /* select fill type user-defined */
```

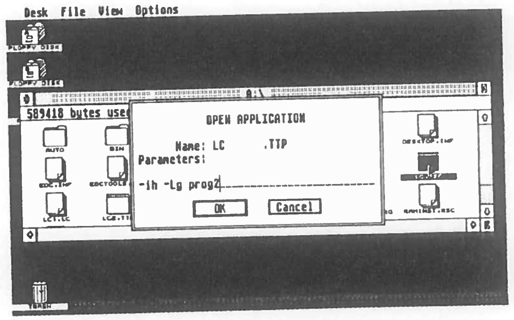
*The PROG2.C Program*

Now compile this program as you did in Lesson 1 but making sure that you select Link with GEM on the Options menu (1Mb+ users) or, if you are a 512K owner, use the -Lg option on the compiler command line (instead of just -L). (TOS 1.0 users will have to use the option -? and then type in the command line as described above).



```
#include <aes.h>          /*
#include <vdi.h>          /*

/* v_opnvwk input array */
short work_in[11]={1,1,1,1

/* v_opnvwk output array *
short work_out[57];

int main(void)
{
    short handle;         /* virtual workstation handle */
    short junk;           /* unused variable */

    appl_init();          /* start AES */
    handle=graf_handle(&junk,&junk,&junk,&junk);     /* find AES handle */
    v_opnvwk(work_in,handle,work_out);               /* open workstation */
    v_clrwk(handle);                                 /* clear workstation */

    vsf_interior(handle;FIS_USER);  /* select fill type user-defined */
```

*Interactively Compiling with GEM*



OPEN APPLICATION
Name: LC .TTP
Parameters:
-ih -Lg prog2
OK Cancel

*512K Users - Compiling with GEM*

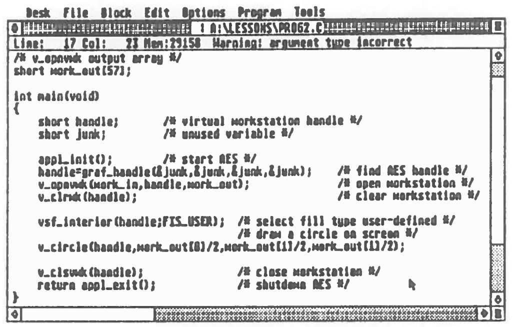
What happened? You should have seen a report something like this:

```
Lattice Atari C Compiler Copyright © 1990 HiSoft & Lattice, Inc.
All rights reserved - Version 5.04.00
                                                              /* open
        v_opnvwk(work_in,handle,work_out);
workstation */
                                  ^
A:\LESSONS\PROG2.c 17 Warning 88: argument type incorrect
        vsf_interior(handle;FIS_USER);  /* select fill type user-defined */
                            ^
A:\LESSONS\PROG2.c 20 Error 16: invalid function argument
        vsf_interior(handle; 4 );       /* select fill type user-defined */
                               ^
A:\LESSONS\PROG2.c 20 Error 57: semi-colon expected

Press any key
```

*The first compilation of PROG2.C*

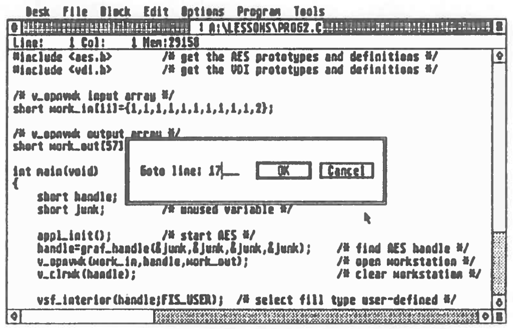
The compiler has generated 1 warning and 2 errors.

Now get back to the editor with the PROG2.C program loaded. To achieve this, 1Mb+ users simply hit a key whilst 512K owners must run EDC.PRG and load PROG2.C. Now, how do we correct these errors? Well, again 1Mb+ users have all the luck - the cursor will already be positioned on the line in which the first error/warning occurred and the description of the problem will be in the top of the window. 512K users should note down the line numbers of the errors/warnings and then use the Goto... command on the Edit menu (or press Alt-G) to position the cursor in the first rogue line.



```
Line:   17 Col:   23 Mem:29158  Warning! argument type incorrect
/* v_opnvwk output array */
short work_out[57];

int main(void)
{
    short handle;       /* virtual workstation handle */
    short junk;         /* unused variable */

    appl_init();        /* start AES */
    handle=graf_handle(&junk,&junk,&junk,&junk);    /* find AES handle */
    v_opnvwk(work_in,handle,work_out);              /* open workstation */
    v_clrwk(handle);                                /* clear workstation */

    vsf_interior(handle;FIS_USER);  /* select fill type user-defined */
                                    /* draw a circle on screen */
    v_circle(handle,work_out[0]/2,work_out[1]/2,work_out[1]/2);

    v_clswk(handle);                /* close workstation */
    return appl_exit();             /* shutdown AES */
}
```

*Interactively Getting to the Error*



```
Line:    1 Col:    1 Mem:29158
#include <aes.h>        /* get the AES prototypes and definitions */
#include <vdi.h>        /* get the VDI prototypes and definitions */

/* v_opnvwk input array */
short work_in[11]={1,1,1,1,1,1,1,1,1,1,2};

/* v_opnvwk output array */
short work_out[57]

int main(void)
{
    short handle;
    short junk;         /* unused variable */

    appl_init();        /* start AES */
    handle=graf_handle(&junk,&junk,&junk,&junk);    /* find AES handle */
    v_opnvwk(work_in,handle,work_out);              /* open workstation */
    v_clrwk(handle);                                /* clear workstation */

    vsf_interior(handle;FIS_USER);  /* select fill type user-defined */
```

*Getting to the Error for 512K Users*

Looking at the first erroneous line (v_opnvwk(work_in, handle, work_out);) it looks legal, but the compiler has given an argument type incorrect warning. This means that the type of one of the arguments in a function call was not what the compiler expected it to be. In fact, the problem here is that the second parameter in the v_opnvwk function has been declared in the header file as a pointer to a short and, as such, the address of handle (&handle) should be passed.

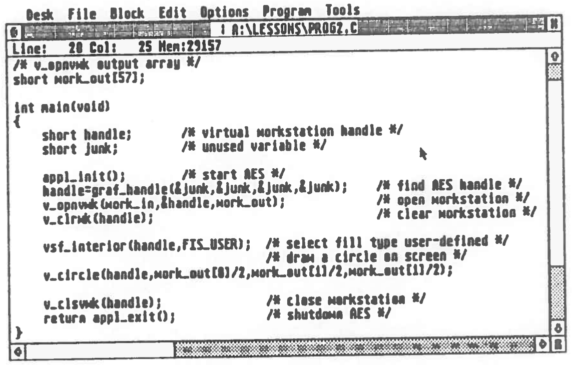
So position the cursor on the h of handle and type an ampersand character, &.

Now go to the next error by pressing Alt-J or Alt-G (depending on your memory - in more ways than one!), this is in the line:

```
vsf_interior(handle;FIS_USER); /* select fill-type */
```

and the error is Invalid function argument. Well, it shouldn't take you too long to spot that the semi-colon between handle and FIS_USER should be a comma. Position the cursor over the F of FIS_USER and hit Backspace followed by a comma - ,.

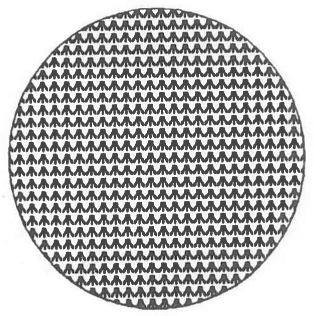
Now compile the program again, remembering to save it first. It should now compile and link successfully, to PROG2.PRG, and you can try running it.



```
Line:   28 Col:   25 Mem:29157
/* v_opnvwk output array */
short work_out[57];

int main(void)
{
    short handle;       /* virtual workstation handle */
    short junk;         /* unused variable */

    appl_init();        /* start AES */
    handle=graf_handle(&junk,&junk,&junk,&junk);    /* find AES handle */
    v_opnvwk(work_in,&handle,work_out);             /* open workstation */
    v_clrwk(handle);                                /* clear workstation */

    vsf_interior(handle,FIS_USER);  /* select fill type user-defined */
                                    /* draw a circle on screen */
    v_circle(handle,work_out[0]/2,work_out[1]/2,work_out[1]/2);

    v_clswk(handle);                /* close workstation */
    return appl_exit();             /* shutdown AES */
}
```

*The Corrected Program*

The program draws a filled circle like this:



*The PROG2.PRG program running*

## Summary of Lesson 2

- Again, 520ST/STE users cannot use a fully integrated system because there is not enough memory to support this - you must note down any errors/warnings and use the Goto line... (Alt-G) feature of the editor.
- 1Mb+ users have an fully-integrated system where the compiler tells the editor about errors and you can use the Jump to Error (Alt-J) command on the Program menu to find all the errors/warnings.
- A sidenote - the warning about argument type incorrect would not have been given by older, non-ANSI, compilers. Lattice C 5 knows about function prototyping and is able to check the types of function arguments.

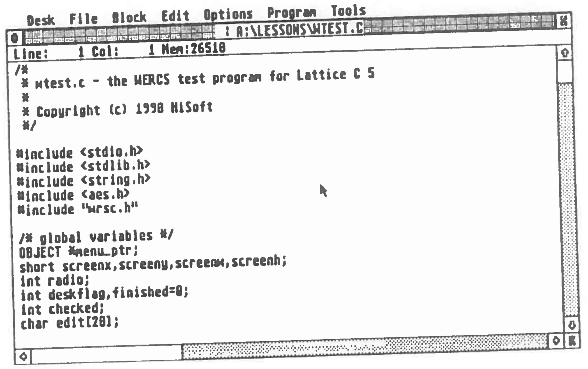
If you are wide awake you may have noticed that the compiler reported two errors (look back at the output on page 13) and that we only corrected one of them to obtain a successful compilation. This is because the second error was caused by the first error; the compiler became confused as to the meaning of the program and thought that a semi-colon was overdue. This is an example of a spurious error caused by a previous problem - always be on the look out for this type of error.

## Lesson 3 - Optionally yours

In this lesson we are not going to differentiate between 512K and 1Mb+ users and we shall simply tell you to compile, link and run your program - please refer to Lesson 1 if you are unsure how to do this.

Run EDC.PRG (512K users) or LC.PRG (1Mb+ users), insert your working disk and type Alt-L or select Load from the File menu

Single-click on the Lessons directory to open it and then double-click on WTEST.C to load it into the editor.



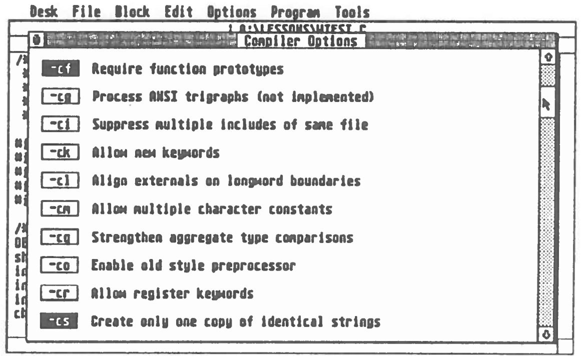
```
/*
 * wtest.c - the WERCS test program for Lattice C 5
 *
 * Copyright (c) 1990 HiSoft
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <aes.h>
#include "wrsc.h"

/* global variables */
OBJECT *menu_ptr;
short screenx,screeny,screenw,screenh;
int radio;
int deskflag,finished=0;
int checked;
char edit[20];
```

*The WTEST.C Program*

This program is an extended example of using the GEM system and uses structures created with the WERCS resource editor - see the chapter **WERCS, The Resource Editor** for more details.

Our concern here is to show you how to compile it using some useful compiler options; these are used to let you modify the way the compiler behaves when compiling your program.

To see most of the compiler options available to you, select Compiler Options... on the Options menu or type Control-O, from within the editor. The following list will appear:



[✓ -cf] Require function prototypes
-cg Process ANSI trigraphs (not implemented)
-ci Suppress multiple includes of same file
-ck Allow new keywords
-cl Align externals on longword boundaries
-cm Allow multiple character constants
-cq Strengthen aggregate type comparisons
-co Enable old style preprocessor
-cr Allow register keywords
[✓ -cs] Create only one copy of identical strings

*The Compiler Options list*

As you can see, from scrolling down the list using the slider on the right hand side, there are a great many such options giving you enormous flexibility in your compilation. The options that are enabled by default are shown highlighted in black.

The options we are going to use in this program are:

-csf this is a multiple option, the -c is a prefix which can be followed by certain other letters to give particular options.

- -cs *create only one copy of identical strings* - this keeps the program smaller by only keeping one copy of any identical constant strings.
- -cf *require function prototypes* - function prototypes were introduced into the C language by the recent ANSI standard and are used to tell the compiler exactly how a function should be called, thus making it easier for you. Lattice C 5 checks for a function prototype and, if this option is used, will warn you if a prototype is absent.

Thus -csf says *create only one copy of identical strings* and *require function prototypes*; you can add other -c options onto this list.

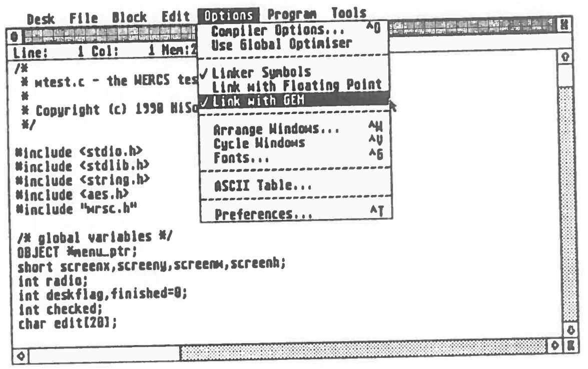
-v *disable stack checking code* - this option tells the compiler not to generate any inline code in your program that will check that there is sufficient stack space during your program's execution. You would normally only use this option when you had a completely finished program that had been fully tested and that you wished to be of a minimum size and to run at maximum speed.

-Lag this is a multiple option, the -L is a prefix which can be followed by certain other letters to give particular options for the linker.

- -La *add symbols to the created program, for debugging purposes* - this adds all your global variable and function names to your program so that a debugger can pick them up and make it easier for you to debug your program.
- -Lg *link with GEM* - this ensures that the linker searches the GEM libraries when resolving references on your program.

Note that both these options are available from the Options menu when compiling interactively, from the editor. When using LC.TTP you would use them on the command line, instead.

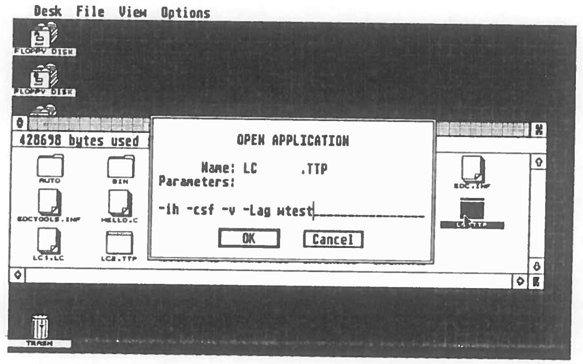
So, if you are running interactively, type Control-O and select -cs, -cf & -v and then go up to the Options menu and ensure that it looks like:



```
/*
 * wtest.c - the WERCS tes
 *
 * Copyright (c) 1990 HiSo
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <aes.h>
#include "wrsc.h"

/* global variables */
OBJECT *menu_ptr;
short screenx,screeny,screenw,screenh;
int radio;
int deskflag,finished=0;
int checked;
char edit[20];
```

*The Editor's Options menu*

Otherwise, if you have less than 1Mb of memory, quit the editor, invoke LC.TTP as in Lesson 1 and type the following command line:



```
OPEN APPLICATION
Name: LC       .TTP
Parameters:
-ih -csf -v -Lag wtest
OK   Cancel
```

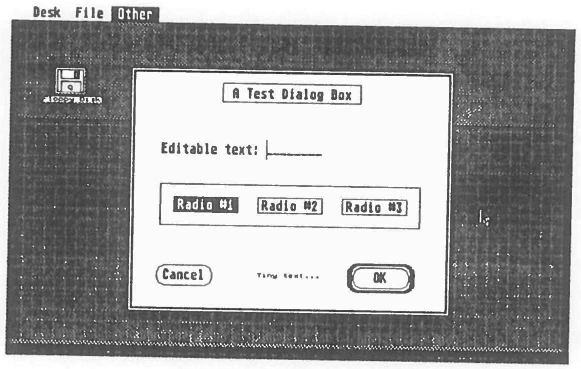
*512K Users - The Command Line for WTEST.C*

Compile and link the program as usual - you will create WTEST.PRG.

1Mb+ users can do this, and run the result, simply using Alt-X.

512K owners will use LC.TTP as described and then double-click on WTEST.PRG from the desktop.

Either way, the running program looks like:



*WTEST.PRG running*

## Summary of Lesson 3

- The Lattice C 5 system is extremely flexible and allows the user great freedom of choice.
- There are many compiler and linker options that affect how these tools will treat your program. These options also often affect your program's size and speed of execution; it is wise to use them carefully and only when you need them.
- It is very easy to create GEM programs with Lattice C 5; once you have experimented a little you might like to read through the WERCS chapter later in this manual which contains much invaluable information regarding GEM. **Volume III** (the Atari Library manual) is worth dipping into for further information on the various aspects of GEM.

That completes our brief but, we hope, useful introduction to using the Lattice C 5 system. We now encourage you to get on and use the package, referring to this and the other volumes as and when you need to although you might like to glance at the final section in this chapter before you do.

## Hints and Tips

Here is some advice on getting the best out of your Lattice C 5 system and your Atari ST.

- Always read the error messages very carefully - they invariably provide much information about the source of the problem. Do not expect the error message to necessarily occur in the line that is actually at fault, the compiler may often report an error one line too late. Also, the error pointer (^) often points after the error, not at it.

- If you are totally lost as to why your program is not working, use the manuals in the first instance - look up the library function definition and check that you are using it correctly and including the right header files. The manuals are full of useful and relevant information, please use them for reference as often as possible.

- It is best if you do not ignore warnings generated by the compiler - it spots many semantic mistakes and gives you guidance, through the warnings, as to errors you may be making in your usage of types.

  In particular, the argument type incorrect warning means that you are passing an expression to a function that is not consistent with what the function expects - *do not just cast the argument* (force its type) to get yourself out of trouble, either change the definition of the function you are calling or change the type of the variable to be appropriate to that expected by the function. See Lesson 2 above.

- If (when!) your program crashes don't be afraid to use the debugger - it is worth getting used to single-stepping and setting breakpoints, not only will you gain understanding of how your program is running but you will also speed up the discovery of obscure runtime errors. See the chapter on MonST2C for more detail.

- If possible, use a RAM disk for the intermediate (quad) files generated by the compiler since this has a massive effect on the speed of compilation. Do this by setting the QUAD environment variable - see the chapter on **LC, the Compiler**.

- If the linker reports undefined symbols and they are not your program's fault, make sure that the first file loaded by the linker is c.o (or the equivalent file required by the library that you are using) i.e. put it first in the linker's command. If you are using more than one library then lc.lib (or the equivalent) should be searched last.

- If you wish to create a linker control (with) file then don't get put off by the many options, get the LC.TTP driver to build it for you by using the -L option, see the **LC, the Compiler** chapter.

- All the advanced features of the Lattice linker (CLink) naturally require that you use Lattice format objects rather than GST format objects. It is always worth changing any existing assembly language code to be able to be assembled by asm so that you may use CLink.

- If you have code that does not use prototypes, perhaps because you haven't used an ANSI compiler before, you can use the -pr option of the compiler to generate a prototype file for your functions, automatically.

- You can turn stack checking on and off for each individual module in a multi-module program. This can be particularly useful if one module of a program is highly recursive.

- If your program has several large static arrays, the best code will be produced if you use the default (small, -b1) data model i.e. don't use the -b0 option. Instead, declare the large arrays as far, explicitly.

- Unlike most 68000 C compilers, Lattice C 5 lets you produce extremely large programs that use the small, fast branch instructions of the 68000 instruction set. Thanks to the advanced *alv* (automatic link vector) facility of the linker these branches are extended when required. The -r0 option overrides this feature making all branches long, for GST compatibility - unless you have a very good reason to do otherwise, do not use the -r0 option; we have never used -r0, even though LC.PRG is nearly 200K long and is compiled with itself.

- To generate small, fast programs, assuming that you are using prototypes, use the -rr option which will cause up to four parameters to be passed in *registers* rather than on the stack which is slower and requires more code. Note that this option may not be used if you are *not* using prototypes.

- Using 16 bit integers as the default (via the -w flag) will give better code than using the default 32 bit integers. However we recommend strongly that you use prototypes with this option, since otherwise it is easy to end up with the wrong number of bytes being placed on the stack when calling a function, often with disastrous results!

- As always, the more memory you have, the easier and the faster will be your development. Memory prices are reasonably low at the time of writing and we would advise you to make sure you have at least one megabyte of memory in your machine as soon as possible!

- You will also find that the purchase of a hard disk drive will revolutionise your attitude to program development, as long as you organise it well and keep regular backups!

Right, now it's up to you!

# EdC
# The Screen Editor

## Introduction

This chapter details the use of the editor, EdC, and how to invoke other parts of the system from it; it does not detail those tools themselves. EdC is an enhanced version of the editor supplied with HISoft DevpacST and HISoft BASIC. In many ways EdC is more than an editor, it is a visually-orientated shell that will let you run almost the whole Lattice C system from within it using a single keystroke or menu click. Having said that you will need either a hard disk or two double-sided floppies to take full advantage of this.

EdC comes in two versions, one that includes the compiler (LC.PRG) and the other that does not (EDC.PRG). They are both used in the same way, but the version with the compiler has extra commands, and uses more memory of course. In the rest of this section EdC refers to those facilities that are available from both EdC and LC; whilst LC refers to those features that are only available from LC.PRG.

To run EdC, double-click on the EDC.PRG icon from the Desktop or type EDC from Batcher. LC.PRG is loaded in the same way.

When the editor has loaded, a menu bar will appear and an empty window will open, ready for you to enter your programs.

## The Editor

The editor section of EdC is a screen editor which allows you to enter and edit text and save and load from disk, as you would expect. It also lets you print some or all of your text, search and replace text patterns and manipulate blocks of text. It is GEM-based, which means it uses all the user-friendly features of GEM programs that you have become familiar with such as windows, menus and mice. However, if you're a die-hard, used to the hostile world of computers before the advent of WIMPs, you'll be pleased to know you can use most of the commands from the keyboard without having to touch the mouse.

The editor is RAM-based, which means that the file you are editing stays in memory for the whole time, so you don't have to wait while your disk grinds away loading different sections of the file as you edit. If you have enough memory you can edit a file of over 300k (though make sure your disk is large enough to cope with saving it if you do!). As all editing operations, including operations like searching, are RAM-based they act very quickly.

When you have typed in your program you must be able to save it to disk, so the editor has a comprehensive range of save and load options, allowing you to save all or part of the text and to load other files into the middle of the current one, for example. It will also let you edit up to four files at once, so that you can check the contents of a header file whilst writing your program.

Features may be accessed in one or more of the following ways:

- Using a single key, such as a Function or cursor key;
- Clicking on a menu item, such as Save;
- Using a menu shortcut, by pressing the Alternate key (subsequently referred to as Alt) in conjunction with another, such as Alt-F for Find;
- Using the Control key (subsequently referred to as Ctrl) in conjunction with another, such as Ctrl-A for cursor word left;
- Clicking on the screen, such as in a scroll bar.

The menu shortcuts have been chosen to be easy and obvious to remember, while the Ctrl commands are based on those used in WordStar, and many other compatible editors since.

## Entering text and Moving about

Having loaded EdC, you will be presented with an empty window with a status line at the top and a flashing black block, which is the *cursor*, in the top left-hand corner.

The status line contains information about the cursor position in the form of line and column offsets as well as the number of bytes of memory which are free to store your text. Initially this is displayed as 29980, as the default text size is 30000 bytes.

You may change this default if you wish, together with various other options, by selecting Preferences, described later. The 'missing' 20 bytes are used by the editor for internal information. The rest of the status line area is used for error messages, which will usually be accompanied by a 'ping' noise to alert you. Any message that is printed will be removed subsequently when you press a key.

### Cursor keys

To move the cursor around the text to correct errors or to enter new characters, you use the cursor keys, labelled ← → ↑ and ↓. If you move the cursor past the right-hand end of the line this won't add anything to your text, but if you try to type some text at that point the editor will automatically add the text to the real end of the line. If you type in long lines the window display will scroll sideways if necessary.

If you cursor up at the top of a window the display will either scroll down if there is a previous line, or print the message Top of file in the status line. Similarly if you cursor down off the bottom of the window the display will either scroll up if there is a following line, or print the message End of file.

You can move the cursor on a character basis by clicking on the arrow boxes at the end of the horizontal and vertical scroll bars.

For those of you used to WordStar style editors, the keys Ctrl-S, Ctrl-D, Ctrl-E and Ctrl-X work in the same way as the cursor keys.

To move immediately to the start of the current line, press Ctrl ←, and to move to the end of the current line press Ctrl →.

To move the cursor a word to the left, press Shift ← and to move a word to the right press Shift →. You cannot move past the end of a line with Shift →. A word is defined as anything surrounded by a space, a tab or a start or end of line. The keys Ctrl-A and Ctrl-F also move the cursor left and right on a word basis.

To move the cursor a page up, you can click on the upper grey part of the vertical scroll bar, or press Ctrl-R or Shift ↑. To move the cursor a page down, you can click on the lower grey part of the scroll bar, or press Ctrl-C or Shift ↓.

If you want to move the cursor to a specific position on the screen you may move the mouse pointer to the required place and click (there is no WordStar equivalent for this feature!).

### Tab key

Pressing the Tab key inserts a special character (ASCII code 9) into your text, which on the screen looks like a number of spaces, but is rather different. Pressing Tab aligns the cursor onto the 'next multiple of 4 column', so if you press it at the start of a line (column 1) the cursor moves to the next multiple of 4, +1, which is column 5.

When you delete a tab the line closes up as if a number of spaces had been removed. The advantage of tabs is that they take up only 1 byte of memory, and only one byte on disk, but can show on screen as many more, allowing you to tabulate your program neatly, without increasing its size unduly. You can change the tab size before or after loading EdC using the Preferences command described shortly.

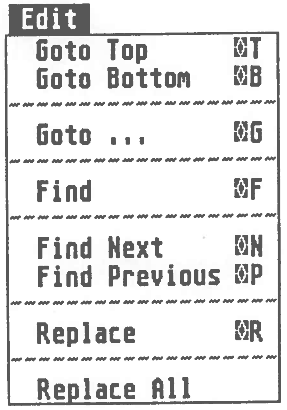
## Backspace key

Pressing the Backspace key removes the character to the left of the cursor. If you backspace at the very beginning of a line it will remove the invisible carriage return and join the line to the end of the previous line. Backspacing when the cursor is past the end of the line will delete the last character on the line, unless the line is empty in which case it will re-position the cursor at the left of the screen.

## Delete key

The Delete key removes the character under the cursor and has no effect if the cursor is past the end of the current line.

The commands on the Edit menu may also be used to move the cursor about your text:



## Goto line

To move the cursor to a specific line in the text, click on Goto ... from the Edit menu, or press Alt-G. A dialog box will appear, allowing you to enter the required line number. Press Return or click on the OK button to go to the line or click on Cancel to abort the operation. After clicking on OK the cursor will move to the specified line, re-displaying if necessary, or give the error End of file if the line doesn't exist.

Another fast way of moving around the file is by dragging the slider on the vertical scroll bar, which works in the usual GEM fashion.

## Go to top of file

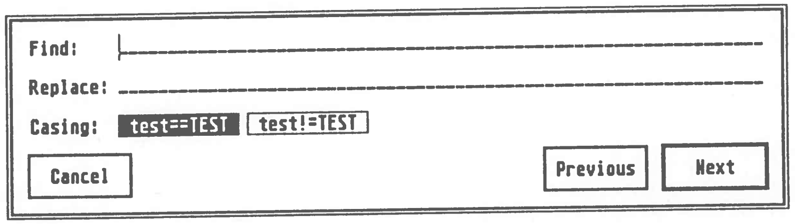
To move to the top of the text, click on Goto Top from the Edit menu, or press Alt-T. The screen will be re-drawn if necessary starting from line 1.

## Go to end of file

To move the cursor to the start of the very last line of the text, click on Goto Bottom, or press Alt-B.

## Searching and Replacing Text

To find a particular section of text click on Find from the Search menu, or press Alt-F. A dialog box will appear,



This allows you to enter the find and replace strings. In the example above long has been entered as the find string and int as the replace string.

If you click on Cancel, no action will be taken; if you click Next (or press Return) the search will start forwards, while clicking on Previous will start the search backwards. If you do not wish to replace, leave the replace string empty.

If the search is successful, the screen will be re-drawn at that point with the cursor positioned at the start of the string. If the string could not be found, the message Not found will appear in the status area and the cursor will remain unmoved.

Whether test is treated as the same as TEST or Test etc. depends on which Casing button is selected. In the example above the search would not stop if LONG was found; if test==Test was selected then the search would find LONG.

To find the next occurrence of the string click on Find Next from the Edit menu, or press Alt-N. The search starts at the position just past the cursor.

To search for the previous occurrence of the string click on Find Previous from the Search menu, or press Alt-P. The search starts at the position just before the cursor.

Having found an occurrence of the required text, it can be replaced with the replace string by clicking on Replace from the Edit menu, or by pressing Alt-R. Having replaced it, the editor will then search for the next occurrence.

If you wish to replace every occurrence of the find string with the replace string from the cursor position onwards, click on Replace All from the Edit menu. During the global replace the Esc key can be used to abort when the status area will show how many replacements were made. There is deliberately no keyboard equivalent for Replace All to prevent it being chosen accidentally.

To search and replace Tab characters press Ctrl-I when typing in the dialog box. Other control characters may be searched for in a similar manner except for the CR (Ctrl-M) and LF (Ctrl-J) characters.

## Deleting text

### Delete line

The current line can be deleted from the text by pressing Ctrl-Y.

### Delete to end of line

The text from the cursor position to the end of the current line can be deleted by pressing Ctrl-Q. (This is equivalent to the WordStar sequence Ctrl-Q Y).

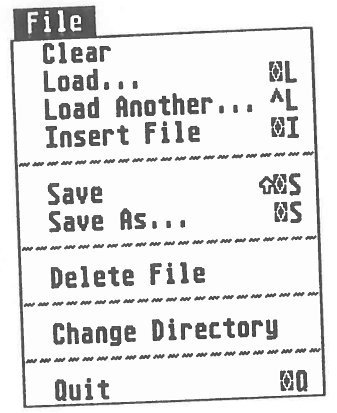
### UnDelete Line

When a line is deleted using either of the above commands it is preserved in an internal buffer, and can be re-inserted into the text by pressing Ctrl-U, or the Undo key. This can be done as many times as required, particularly useful for repeating similar lines or swapping over individual lines.

### Delete all text

To clear out the current text, click on Clear from the File menu. If you have made any changes to the text that have not been saved onto disk, a confirmation is required and an alert box will appear. Click on OK to delete the text, or on Cancel to abort the operation. If you wish to save the text before clearing the buffer, click on the Save button.

## Disk Operations



### Save As...

To save the text you are editing, click on Save As... from the File menu, or press Alt-S. The GEM File Selector will appear, allowing you to select a suitable disk and filename. Clicking OK or pressing Return will then save the file onto the disk. If an error occurs a dialog will appear showing a TOS error number, the exact meaning of which can be found under _OSERR in **Volume II - Library manual**.

If you click on Cancel the text will not be saved.

Normally if a file exists with the same name it will be deleted and replaced with the new version, but if Backups are selected from the Preferences options then any existing file will be renamed with the extension .BAK (deleting any existing .BAK file) before the new version is saved.

### Save

If you have already done a Save As (or a Load), EdC will remember the name of the file and display it in the title bar of the window. If you want to save it without having to bother with the file selector, you can click on Save on the File menu, or press Shift-Alt-S, and it will use the old name and save it as above. If you try to Save without having previously specified a filename you will be presented with the File Selector, as in Save As.

### Loading Text

To load in a new text file, click on Load from the File menu, or press Alt-L. If you have made any changes that have not been saved, a confirmation will be required. The GEM file selector will appear, allowing you to specify the disk and filename. Assuming you do not Cancel, the editor will attempt to load the file. If it will fit, the file is loaded into memory and the window is re-drawn. If it will not fit an alert box will appear warning you, and you should use Preferences to make the edit buffer size larger, then try to load it again.

If the file can't be found then a dialog box will appear, asking you if you wish to create that file. You may do so, or alternatively modify the filename and try again.

If you wish to continue editing the current file and would like to edit another file then use Load Another... or press Ctrl-L. This will open the file in the next unused window.

When loading EdC from Batcher, or any other CLI, you may include up to four filenames. The corresponding files will then be loaded automatically. If a file cannot be found you will be asked if you wish to create it or may change the filename if you wish.

## Inserting Text

If you want to read a file from disk and insert it at the current position in your text click on Insert File from the File menu, or pressAlt-I. The GEM file selector will appear and assuming that you do not cancel, the file will be read from the disk and inserted, memory permitting.

## Delete File

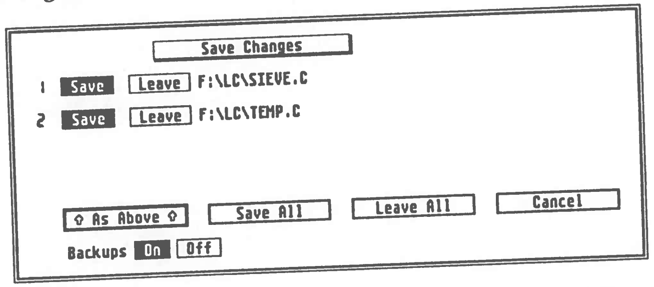
If you want to delete a file on disk (if for instance you have run out of disk whilst trying to save), click on Delete File. The GEM file selector will appear, allowing you to select a suitable disk and filename. Clicking OK or pressing Return will then delete the file from the disk. If an error occurs a dialog will appear showing a TOS error number, the exact meaning of which can be found under _OSERR in **Volume II - Library manual**. If you click on Cancel the file will *not* be deleted.

## Change Directory

This option allows you to move the current directory; this can be useful when running programs which expect all of their files to be in the same place as the program itself. After clicking on Change Directory the GEM file selector will appear, allowing you to select a suitable disk and folder name. Clicking OK or pressing Return will then change the directory. If you click on Cancel the directory will not be changed.

## Quitting EdC

To leave EdC, click on Quit from the File menu, or press Alt-Q. If changes have been made to the text which have not been saved to disk, an alert box will appear asking for confirmation, like this:

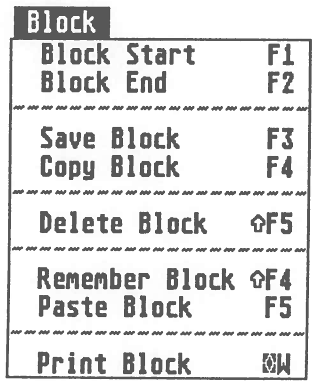


This example shows that two files have changed. Clicking on Save All, As Above or pressing Return will exit the editor saving the changes. Clicking on Cancel will return to the editor. Leave all will ignore all the changes you have made.

If you wish to save some files but not others click on the appropriate Leave buttons. For example if you clicked on the Leave button by F:\LC\TEMP.C in the above example and then pressed Return, only the F:\LC\SIEVE.C file would be saved.

You can also enable and disable backups from this dialog box. This is useful if you normally use backups, but decide that you don't require a backup of a one line change.

## Block Commands



```
Block
 Block Start      F1
 Block End        F2
 Save Block       F3
 Copy Block       F4
 Delete Block    ⇧F5
 Remember Block  ⇧F4
 Paste Block      F5
 Print Block     ⇧W
```

A *block* is a marked section of text which may be copied to another section, deleted, printed or saved onto disk. The function keys are used to control blocks.

### Marking a block

The start of a block is marked by moving the cursor to the required place and selecting Block Start or pressing key F1. The end of a block is marked by moving the cursor and selecting Block End or pressing key F2. The start and end of a block do not have to be marked in a specific order - if it is more convenient you may mark the end of the block first.

A marked block is highlighted by showing the text in reverse. While you are editing a line that is within a block this highlighting will not be shown but will be re-displayed when you leave that line or choose a command.

### Saving a block

Once a block has been marked, it can be saved by clicking on Save Block from the Block menu or by pressing key F3. If no block is marked, the message What blocks! will appear. If the start of the block is textually after its end the message Invalid block! will appear. Both errors abort the command. Assuming a valid block has been marked, the GEM file selector will appear, allowing you to select a suitable disk and filename. If you save the block with a name that already exists the old version will be overwritten - no backups are made with this command.

### Copying a block

A marked block may be copied, memory permitting, to another part of the text by moving the cursor to where you want the block copied and clicking on Copy Block or by pressing key F4. If you try to copy a block into a part of itself, the message Invalid block! will appear and the copy will be aborted.

### Deleting a block

A marked block may be deleted from the text by clicking on Delete Block or by pressing Shift-F5. The shift key is deliberately required to prevent it being used accidentally. A deleted block is remembered, memory permitting, in the block buffer, for later use.

### Copy block to block buffer

The current marked block may be copied to theblock buffer, memory permitting, using Remember Block or by pressing Shift-F4. This can be very useful for moving blocks of text between different files by loading the first, marking a block, copying it to the block buffer then switching to another window or loading the other file and pasting the block buffer into it.

### Pasting a block

A block in the block buffer may be pasted at the current cursor position by clicking on Paste Block or by pressing F5.

Note The block buffer will be lost if the edit buffer size is changed.

### Printing a block

A marked block may be sent to the printer by clicking on Print Block or by pressing Alt-W. An alert box will appear confirming the operation and clicking on OK will print the block. The printer port used will depend on the port chosen with the Install Printer desk accessory, or will default to the parallel port. Tab characters are sent to the printer as a suitable number of spaces, so the net result will normally look better than if you print the file from the Desktop.

Block markers remain during all editing commands, moving where necessary, and are only reset by the commands Clear, Delete block, and Load.

Note If you try to print when no block is marked at all then the whole file will be printed.