# Installing and Running Let's C

This section describes how to install **Let's C** onto your computer, and how to use it to compile simple programs.

## Installing Let's C

Before you can use **Let's C**, you must *install* it on your computer. As **Let's C** comes to you, its files are fitted together to save space on its disks, not to run conveniently. This helps us lower our costs, to give you **Let's C** at the lowest possible price; however, it also means that before you can use the compiler you must recopy the files into organized groups.

To install **Let's C**, you must copy files from the distribution disks onto either a hard disk or, if you don't have a hard disk, a set of floppy disks. To make this job easy for you, **Let's C** includes the utility **install**, which does the copying for you. By running **install** and answering a few simple questions, you can build a working copy of **Let's C** on your system in a few minutes.

If you have a hard disk, use the directions in the section *Installing Let's C onto a hard disk*. If you do not have a hard disk, skip below to the section *Installing Let's C onto a floppy-disk system*, and follow the directions there.

### Installing Let's C onto a hard disk

To begin, log into drive C on your system. You can do so by typing **c** at the MS-DOS prompt. On nearly all computers, drive C is the hard disk.

Now, insert **Let's C**'s distribution disk 1 into floppy drive A and type the following command:

```
a:install
```

In a moment, **install** will begin to work. It will print some information on your screen, and then ask you the following question:

```
Do you have a hard disk?
```

Type 'y', for "yes".

**install** will then ask you:

```
Do you wish to install all the files?
```

Answer 'y'.

**install** will now ask you in which directories you wish to install the files, as follows:

```
Where do you want the executable programs?
Where do you want the header files?
Where do you want the libraries?
Where do you want the sample programs?
Where do you want the source files?
Where do you want the temporary files?
```

After each question, type **<return>**, which signifies the default setting. Later, you may wish to re-install **Let's C** into other directories, but you should use the default settings until you gain some familiarity with **Let's C**.

**install** will now copy the files from the distribution disk onto your hard disk. Depending on how fast your system is and how fast your hard disk is, this can take from five to 15 minutes. When all the files from one disk are copied, **install** will ask for the next disk, until all files are copied.

When **install** exits, it will print some information on the screen, and then an instruction of the form:

```
set CCHEAD=@c:\lib\ccargs
```

Copy down this instruction. You should then write this instruction into the file **autoexec.bat** on your MS-DOS boot disk. This instruction tells **Let's C** where you have stored all of its components, so it can find everything correctly. This will be discussed below, in the section entitled *Setting your computer's environment*.

Once all the files are copied, place your copy of the MS-DOS disk into the floppy disk drive. If you have version 3.2 of MS-DOS, you have two MS-DOS disks; insert the disk labelled "supplemental programs". Now, type the command:

```
copy a:link.exe \bin
```

This copies the MS-DOS linker MS-LINK into directory **\bin** on your hard disk. **Let's C** uses MS-LINK to link its executable files, so MS-LINK must be copied into a directory where **Let's C** can find it. If you did not use the default directory names, copy **link.exe** into the directory where you stored **Let's C**'s executable files.

That's all there is to it. **Let's C** is now installed on your hard disk. Now, skip below to the section entitled *Setting your computer's environment*. This will tell you how to set the environment, so **Let's C** can work efficiently.

### Installing Let's C onto a floppy-disk system

**Let's C** is too large to fit onto a single floppy disk. Therefore, if your system does not have a hard disk you must install **Let's C** onto a set of seven floppy disks, as follows:

**Disk 1** The **shell** disk. This disk holds MWS, the Mark Williams shell. You will use it only to boot MWS, then put it away.

**Disk 2** The compiler disk for standard-sized programs. You should use this disk to compile programs that are of normal size and complexity. This disk holds the compiler controller **cc**, the phases of compilation, the MicroEMACS screen editor, and other tools used during compilation.

**Disk 3** The compiler disk for unusually large programs. If you have written a program thatis unusually large or complex, and it will not compile correctly with disk 2, use disk 3 instead. This disk has all of the files that appear on disk 2, plus LARGE-model versions of the compiler phases **cc0** and **cc2**.

**Disk 4** This disk holds the SMALL-model libraries. You will need to copy MS-LINK onto this disk.

**Disk 5** This disk holds the LARGE-model libraries. You will also need to copy MS-LINK onto this disk.

**Disk 6** This disk holds the **Let's C** commands and utilities. For a fuller description of the commands, see the Lexicon entry for **commands**.

**Disk 7** This disk holds the sample programs and source code that comes with **Let's C**. This includes the full source code for the MicroEMACS screen editor.

To begin, format eight new floppy disks. Label them respectively as follows:

```
1. shell
2. normal compiler
3. compiler for large programs
4. SMALL-model libraries
5. LARGE-model libraries
6. commands
7. source code and samples
```

Now, at the MS-DOS prompt, type **B:**. This will log into drive B on your machine. Insert **Let's C**'s distribution disk 1 into drive A; type the following command:

```
a:install
```

In a moment, **install** will begin to execute, and will print some information on your screen. It will then ask you this question:

```
Do you have a hard disk?
```

Answer 'n', for "no".

**install** will then ask:

```
Do you wish to install all of the files?
```

Type 'y', for "yes".

**install** will now ask you in which directories you wish to install the files, as follows:

```
Where do you want the executable programs?
Where do you want the header files?
Where do you want the libraries?
Where do you want the sample programs?
Where do you want the source files?
Where do you want the temporary files?
```

After each question, type **<return>**, which accepts the default setting. Later, you may wish to re-install **Let's C** into other directories, but you should use the default settings until you gain some familiarity with **Let's C**.

**install** will now tell you:

```
Insert the shell disk into drive B.
```

Insert the formatted floppy disk that you labelled "shell" into drive B. **install** will copy the appropriate files onto it. When **install** needs a new source disk, it will prompt you for it.

**install** will go through this procedure for each of the seven floppy disks that you will be building. It will prompt you when to change disks, and tell you which disk to insert into drive A or drive B.

When **install** exits, it will print some information on the screen, and then an instruction of the form:

```
set CCHEAD=@a:\ccargs
```

Copy down this instruction. You should then edit this instruction into the file **autoexec.bat** on your MS-DOS boot disk. This instruction tells **Let's C** where you have stored all of its components, so it can find everything correctly. This will be discussed below, in the section entitled *Setting your computer's environment*.

When **install** has finished, you must do the following for each of the four library disks, disks 3 through 6. First, place your computer's MS-DOS disk into drive A. If you have MS-DOS version 3.2 or later, your copy of MS-DOS comes on two disks; insert the disk labelled "supplemental programs" into drive A. Then, place disk 4 (which holds the SMALL-model libraries) into drive B. Type the following command:

```
copy a:link.exe b:\bin
```

This command will copy the MS-DOS linker MS-LINK into directory **\bin** on your library disk. **Let's C** uses MS-LINK to link the executable files it creates, so MS-LINK must be copied into a directory where **Let's C** can find it.

Repeat this procedure for disk 5.

That is all there is to it: **Let's C** is now installed on your computer.

When you are finished, you may wish to recopy your installed disks, to save yourself the trouble of having to reinstall should something happen to your working copy.

Now, read the following section *Setting your computer's environment*, which tells you how to set your computer's environment so you can use **Let's C**.

### Re-installing a portion of Let's C

If you wish, you can re-install just a portion of the compiler. When **install** asks

```
Do you wish to install all of the files?
```

answer 'n'. **install** will then prompt you for which portion, or portions, of **Let's C** you wish to install, and will then install it for you.

## Setting your computer's environment

As you have probably noticed by now, **Let's C** is not just one program: it is a collection of more than 100 files and programs, which together form one of the most sophisticated C programming systems available at any price. Despite its complexity, **Let's C** is designed to work smoothly, and transform your source code into an executable file in the shortest possible time.

For **Let's C** to work at peak efficiency, however, you should pay some attention to your computer's *environment*. The environment is a set of variables that are available to all of the programs run on your computer. By setting the environment properly, you will help **Let's C** find all of its programs quickly to speed your work.

### Setting the PATH

When you installed **Let's C**, all of its commands were copied into a directory called **\bin**. When you type a command into MS-DOS, MS-DOS looks for it in the directories named in the environmental variable **PATH**. To ensure that MS-DOS can find **Let's C**'s commands you must set the **PATH** so that it names the directory **\bin**.

To set the **PATH**, use the MS-DOS command **path**. For example, if you want keep your executable files in the directories **bin** and **mwc**, type:

```
PATH=C:\BIN;C:\MWC
```

If your computer does not have a hard disk, use the following form of the **path** command:

```
PATH=A:\BIN;A:\MWC;B:\BIN;B:\MWC
```

This tells MS-DOS to look for **cc** in the directories **bin** and **mwc** on both of your floppy disk drives. With the **PATH** set to this configuration, it does not matter which disk drive holds your compiler disk when you work: MS-DOS will find **cc** in either drive automatically.

When the **PATH** is set properly, you should copy the **path** command you used into the file **autoexec.bat**, on your computer's boot disk. Then, the **PATH** will be set automatically whenever you boot your computer.

### Finding the ccargs file

When you installed **Let's C** onto your computer, the program **install** created a file called **ccargs**. This file contains the names of all of the directories in which **install** stored the elements of **Let's C**.

When you compile a program, **Let's C** reads **ccargs** and uses its entries to find all of the files it needs. If you have a floppy-disk system, **install** wrote **ccargs** into directory **a:** on disk 2 (the compiler disk). If you have a hard disk, **install** wrote **ccargs** into the directory in which you installed the **Let's C** libraries (the default is **c:\lib**).

You must set an additional environmental variable so that **Let's C** can locate **ccargs** when you compile a program: the environmental variable **CCHEAD**. As noted above, when **install** finished installing **Let's C** on your system, it printed on your screen an instruction of the form

```
set CCHEAD=@directory\ccargs
```

where *directory* is the name of the directory into which it wrote **ccargs**. For example, if you have a floppy-disk system, **install** printed the message

```
set CCHEAD=@a:\ccargs
```

whereas if you have a hard disk, it printed the message

```
set CCHEAD=@c:\lib\ccargs
```

Be sure that you copy this instruction into the file **autoexec.bat** on your MS-DOS boot disk. Once you have done this, reboot your system; this ensures that **CCHEAD** is set for your system.

That's all there is to it. When you copy the **set** commands for **PATH** and **CCHEAD** into **autoexec.bat** and reboot your system, you have set **Let's C**'s environment. **Let's C** is now ready to start working for you.

### Editing ccargs

**ccargs** correctly describes where everything is stored on your computer. The default **ccargs** for a hard disk reads as follows:

```
-xcc:\bin\
-xlc:\lib\
-xtc:\tmp\
-Ic:\include\
```

The default **ccargs** for a floppy disk system reads as follows:

```
-xca:\bin\
-xla:\lib\
-Ia:\include\
-Z
```

If you later decide to move part of **Let's C** from one directory to another, you must edit **ccargs** to reflect this change, or **Let's C** will not know where you moved **Let's C**. You should edit **ccargs** as follows:

*Executable files*

If you move the executable files from where you installed them, change the line in **ccargs** that begins **-xc**. For example, if you have a floppy disk system and you move the executable files from directory **\bin** to directory **\mwc**, change the line

```
-xca:\bin\
```

to read

```
-xca:\mwc\
```

*Libraries*

If you move the libraries from where you installed them, you should change the line in **ccargs** that begins **-xl**. For example, if you have a hard disk and you move the libraries from directory **\lib** to directory **\library**, change the line

```
-xlc:\lib\
```

to read

```
-xlc:\library\
```

*Header files*

If you move the header files from where you installed them, you should change the line in **ccargs** that begins **-I**. For example, if you have a floppy disk system and you move the header files from directory **\include** to directory **\header**, change the line

```
-Ia:\include\
```

to read

```
-Ia:\header\
```

*Temporary files*

Finally, **ccargs** tells **Let's C** where to write your temporary files. This is set only in the version of **ccargs** that is used with a hard disk. If you decide to change where **Let's C** writes its temporary files, you must edit the line in **ccargs** that begins **-xt**. For example, if you want to write temporary files into directory **nowhere**, change the line

```
-xtc:\tmp\
```

to read

```
-xtc:\nowhere\
```

If you need more information on where **Let's C** looks for files, or how **Let's C** works in general, look at the Lexicon entry for **cc**. This will describe in some detail how **Let's C** works, and also describe other ways that you can change how **Let's C** looks for its files.

## Using MWS, the Let's C command interface

**Let's C** includes an interface program, MWS, which is designed to help you develop programs more quickly and efficiently. MWS, which stands for "Mark Williams shell", accelerates the speed with which your programs work, and it has a display interface that helps you build commands with ease.

To invoke MWS, simply do the following. If you do not have a hard disk, insert disk 1 (the "shell" disk) into disk drive A. Then, at the MS-DOS prompt type:

```
MWS
```

In a moment, the screen will clear and the MWS main menu will appear:

```
Let's C Version 4.0
(c) 1987 Mark Williams Company, Chicago

+----------------+
|  Edit          |
|  Compile       |
|  Run           |
|  Debug         |
|  Make          |
|  Buffers       |
|  Quick DOS     |
|  !DOS Escape   |
|  New directory |
+----------------+









<return> select      <F1> more help
<→> use arrow keys   <esc> exit menu
```

As you can see, the entry for **Edit** on the menu is marked by a reverse-video band, called the *cursor bar*. In this tutorial, the cursor bar will be shown as shading. The cursor bar indicates the entry in the main menu you wish to select. Try pressing the down-arrow key (↓) on the keypad. The cursor bar now covers the entry **Compile**. Each selection will be discussed in detail in the following sub-sections.

Note that if you do not want to bother with moving the cursor bar, you can pick an option simply by typing its first letter. For example, you can begin to edit a file simply by typing 'e'.

At the bottom of the screen are listed the commands that you can give MWS. As noted above, pressing the arrow keys moves the cursor bar up and down the menu. Pressing **<return>** tells MWS to select the menu entry that the cursor bar is highlighting. Pressing **<esc>** exits. If you are in a sub-menu, pressing **<esc>** returns you to the previous menu; whereas if you are already in the main MWS menu, pressing **<esc>** exits you from MWS altogether, and returns you to MS-DOS.

Finally, pressing the function key **<F1>** prints a help message. Each screen has its own help message, to guide you through MWS even if you do not have this manual available.

### *Editing a file*

MWS includes a full-featured screen editor, called MicroEMACS. An *editor* is a program that lets you type text into your computer, store it on disk, then recall it from disk and change it. You will use an editor to type all of the programs that you compile with **Let's C**.

MicroEMACS allows you to divide the screen into sections, called *windows*, and display and edit a different file in each one. It has a full search-and-replace function, allows you to define keyboard macros, and has a large set of commands for killing and moving text. Also, MicroEMACS has a full help function for C programming. Should you need information about any macro or library function that is included with **Let's C**, all you need to do is move the text cursor over that word and press a special combination of keys; MicroEMACS will then open a window and display information about that macro or function.

**Let's C** includes both a compiled, binary version of MicroEMACS that is ready to use, and the full source code. We invite you to examine the code, modify it, and enhance MicroEMACS to suit your preferences.

For a list of the MicroEMACS commands, see the Lexicon entry for **me**, the MicroEMACS command. A following section of this introduction gives a full tutorial on MicroEMACS. In the meantime, however, you can begin to use MicroEMACS by learning a half-dozen or so commands.

MWS lets you access the MicroEMACS screen directly through its display interface. To edit a file through MWS, do the following. First, if you do not have a hard disk, remove the "shell" floppy disk from drive A and put it aside. Place disk 2, the "compiler" disk, into drive A. Then, press the up-arrow key (↑) until the cursor bar covers the entry **Edit**. Press **<return>**. This *selects* the edit feature; that is, it tells MWS that you wish to use the MicroEMACS editor to edit a file.

In a moment, MWS redraws the screen as follows:

```
Edit

      +======================================+
      | me                                   |
      +======================================+

      +============+
      |  Execute   |
      |  Files     |
      |  New File  |
      +============+









      <return> select      <F1> more help
      <→> use arrow keys   <esc> exit menu
```

The cursor bar is now positioned over the selection **Execute**. Press the ↓ key two times, so the cursor bar is positioned over **New File**. Press **<return>**. The menu disappears, and MWS prompts you to enter the name of the file you wish to create.

Type **hello.c**. Note that the name **hello.c** appears in the long box at the top of the screen; this box is called the *command box*, because it is where MWS builds your command. Note, too, that the cursor bar has returned to the entry **Execute** on the menu.

Now press **<return>**. This tells MWS to execute the command that is displayed in the command box. The screen again clears: you have just invoked the MicroEMACS editor to edit the file **hello.c**.

Now, type the following text, as it is shown here. If you make a mistake, simply backspace over it and type it correctly; the backspace key will wrap around lines:

```
main()
{
        printf("hello, world\n");
}
```

When you have finished, *save* the file by typing **<ctrl-X><ctrl-S>** (that is, hold down the control key and type 'X', then hold down the control key and type 'S'). MicroEMACS will tell you how many lines of text it just saved. Exit from the editor by typing **<ctrl-X><ctrl-C>**.

## Let's C

When you have exited from MicroEMACS, MWS redisplays the edit menu, with the cursor bar positioned over **Execute**. Note that MWS remembers the last command you built for each of the commands on the main menu (that is, the last **Edit** command, the last **Compile** command, and so on). If you do not wish to build a new command, you can re-execute the last command you built by simply pressing **<return>**.

Now, type **<return>**. MWS will invoke MicroEMACS with the last file you edited, **hello.c**. The text of the file you just typed is now displayed on the screen. Try changing the word **hello** to **Hello**, as follows: First, type **<ctrl-N>** That moves you to the *next* line. (The command **<ctrl-P>** would move you to the *previous* line, if there were one.) Now, type the command **<ctrl-F>**. As you can see, the cursor moved *forward* one space. Continue to type **<ctrl-F>** until the cursor is located over the letter 'h' in **hello**. If you overshoot the character, move the cursor *backwards* by typing **<ctrl-B>**.

If you prefer, you can also move the cursor by pressing the arrow keys.

When the cursor is correctly positioned, delete the 'h' by typing the *delete* command **<ctrl-D>**; then type a capital 'H' to take its place.

With these few commands, you can load files into memory, edit them, create new files, save them to disk, and exit. This just gives you a sample of what MicroEMACS can do, but it is enough so that you can begin to do real work.

Now, again *save* the file by typing **<ctrl-X><ctrl-S>**, and exit from MicroEMACS by typing **<ctrl-X><ctrl-C>**. Again, the screen shows the **Edit** screen. Type **<esc>**; as shown at the bottom of the screen, this will exit you from the **Edit** menu, and so return you to the main menu.

Just as a reminder, the following table gives the MicroEMACS commands presented above:

| | |
|---|---|
| **<ctrl-N>** or ↓ | Move cursor to the *next* line |
| **<ctrl-P>** or ↑ | Move cursor to the *previous* line |
| **<ctrl-F>** or → | Move cursor *forward* one character |
| **<ctrl-B>** or ← | Move cursor *backward* one character |
| **<ctrl-D>** | *Delete* a character |
| **<ctrl-X><ctrl-S>** | *Save* the edited file |
| **<ctrl-X><ctrl-C>** | Exit from MicroEMACS |

### Simple compiling

Now that you have prepared a C program with the MicroEMACS editor, the next step is to compile it into executable form.

To compile a program under **Let's C**, you must use the command **cc**. MWS's display interface lets you easily construct commands for **cc** to execute. To see how this works, press the ↓ key once, so the cursor bar is positioned over **Compile**. Press **<return>**, to select this option.

MWS redraws the screen so it appears as follows:

```
Compile
      +----------------------------------------+
      | cc                                     |
      +----------------------------------------+

      +------------+
      |  Execute   |
      |  Options   |
      |  Files     |
      +------------+









      <return> select      <F1> more help
      <→> use arrow keys   <esc> exit menu
```

The cursor bar is positioned over **Execute**. Press the ↓ key, to move the cursor bar to **Options**; then press **<return>**. The screen will be redrawn to appear as follows:

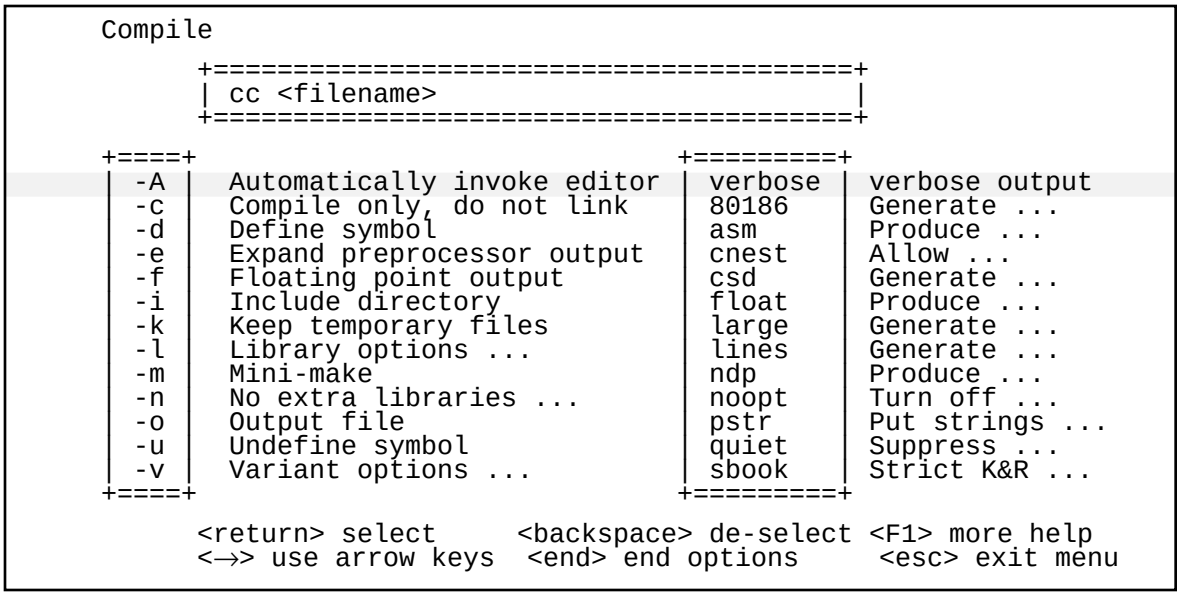
```
Compile
      +========================================+
      | cc <filename>                          |
      +========================================+

+====+
| -A |  Automatically invoke editor
| -c |  Compile only, do not link
| -d |  Define symbol
| -e |  Expand preprocessor output
| -f |  Floating point output
| -i |  Include directory
| -k |  Keep temporary files
| -l |  Library options ...
| -m |  Mini-make
| -n |  No extra libraries ...
| -o |  Output file
| -u |  Undefine symbol
| -v |  Variant options ...
+====+

      <return> select     <backspace> de-select <F1> more help
      <→> use arrow keys <end> end options     <esc> exit menu
```

**cc**'s options cannot all fit onto one screen; if you press the ↓ key until it is at the bottom of the menu, you can then scroll through the options that are not initially shown on the screen. The options with ellipses "…" after their descriptions have a menu of sub-options associated with them.

For the present exercise, type the ↓ key until the cursor bar is positioned over the entry **-v**, for *variant* options. Press **<return>**. A second set of options appears on the screen, which now appears as follows:




The cursor bar is over the **verbose** option. Type **<return>**; this selects the **verbose** option, which tells you what steps the compiler is executing as it compiles. The command box now reads:

```
cc -V <filename>
```

Type **<end>**. The variant options box disappears, and the cursor is repositioned over the **-v** option. Type **<end>** again. The options box has disappeared, and you are back in the **Compile** menu.

Now, press the ↓ key, to position the cursor bar over **Files**. Press **<return>**. In a moment, two boxes will appear. One displays all of the files that have the suffix **.c**, and the other displays all of the files that have the suffix **.obj**, if any. To move between the boxes, press **<tab>**, as shown at the bottom of the screen. Press the ↓ until the cursor bar is positioned over **hello.c**. Press **<return>**. This completes the construction of the **cc** command line, as shown in the command box.

Note that you must select a file by moving the cursor bar to its name and pressing return; you cannot select a file by typing the first character in its name, because more than one file could use that character.

Press **<end>** to tell MWS that you have selected all of the files you want. MWS then returns you to the compile menu, with the cursor bar positioned over **Execute**. Press **<return>**, to begin execution of the command you have built.

Compilation now begins automatically. The switch **-V** tells **cc** to describe each step in compilation.

If your computer does not have a hard disk, **Let's C** will print the following prompt on your screen when it comes time to link your program:

```
Insert floppy disk 3 (SMALL-model, i8087-sensing libraries)
```

Open drive A, remove the compiler disk, and insert floppy disk 3. If your computer does have a hard disk, you will not need to do this. As noted earlier, when you installed **Let's C**, this disk holds the libraries for SMALL-model programs that sense the presence of the i8087 co-processor. For more information on how these libraries work, see the Lexicon entries for **library**, **model**, and **i8087**. When you have inserted this disk, press **<return>**. **Let's C** will now invoke MS-LINK and link your program.

As you can see, **cc** guides the program through all phases of compilation, invokes the linker, and links in all necessary routines from the libraries to produce a file called **hello.exe** that is ready to execute.

When compilation is finished, you will be returned to the MWS main menu.

### Running a program

The next step is to run the program you just compiled. MWS lets you run any program you have compiled with **Let's C**, or in fact any executable program whatsoever, through its display interface.

If your computer does not have a hard disk, before you begin this step you should open drive A, remove floppy disk 3 (a libraries disk), and reinsert floppy disk 2 (the compiler disk). If your computer does have a hard disk, you do not need to do this.

To run **hello.exe**, which you created when you compiled **hello.c**, press the ↓ key until the cursor bar is positioned over **Run**. Then, press **<return>** to select this option. The main menu disappears, and MWS redraws the screen as follows:

```
Run
      +========================================+
      | <filename>                             |
      +========================================+

      +============+
      |  Execute   |
      |  Arguments |
      |  Files     |
      +============+









      <return> select      <F1> more help
      <→> use arrow keys   <esc> exit menu
```

As before, you can use the arrow keys to move the cursor bar up and down the menu. If you press **<return>** with the cursor positioned over **Execute**, MWS will re-execute the last command you built, if any. If your program needs arguments, select the **Arguments** option, which will prompt you to enter them. If your command does not take any arguments, simply type **<return>**.

Press the ↓ key once. The cursor bar is now positioned over **Files**. Press **<return>**. MWS draws a box that holds all executable files in the current directory—that is, all files that have the suffix **.exe**. Press the ↓ key until the cursor bar covers **hello.exe**. Press **<return>**. As you can see, the command box now shows **hello.exe**. Press **<esc>** to exit from this menu.

You have returned to the **Run** menu, with the cursor bar over **Execute**. Press **<return>**. MWS now executes the command shown in the command box, **hello.exe**. It executes, and prints on your screen **hello, world**. To return to MWS's main menu, press any key.

As you can see, it is easy to create, compile, and execute a program through the MWS menu interface.

## Let's C

### Quick DOS and !DOS options

MWS gives you two ways to give commands directly to MS-DOS while still enjoying MWS's acceleration of your programs.

To give just one command to MS-DOS, press the ↓ key until the cursor bar is positioned over **Quick DOS**. Press **<return>**. At the bottom of your screen appears the prompt

```
DOS command:
```

Type **dir**. MS-DOS executes **dir** and prints the contents of the current directory on your screen. When **dir** has finished executing, press any key to return to MWS's main menu.

If you want to give a number of commands directly to MS-DOS, press the ↓ key until the cursor bar is positioned over **!DOS Escape**. Press **<return>**. The main menu again disappears, and your MS-DOS prompt appears on the screen. MS-DOS is now ready to receive any number of your commands. If you are familiar with the **cc** command or MicroEMACS, you can now type these commands directly to MS-DOS. MWS does not require you to use its display interface, but it will still accelerate your work.

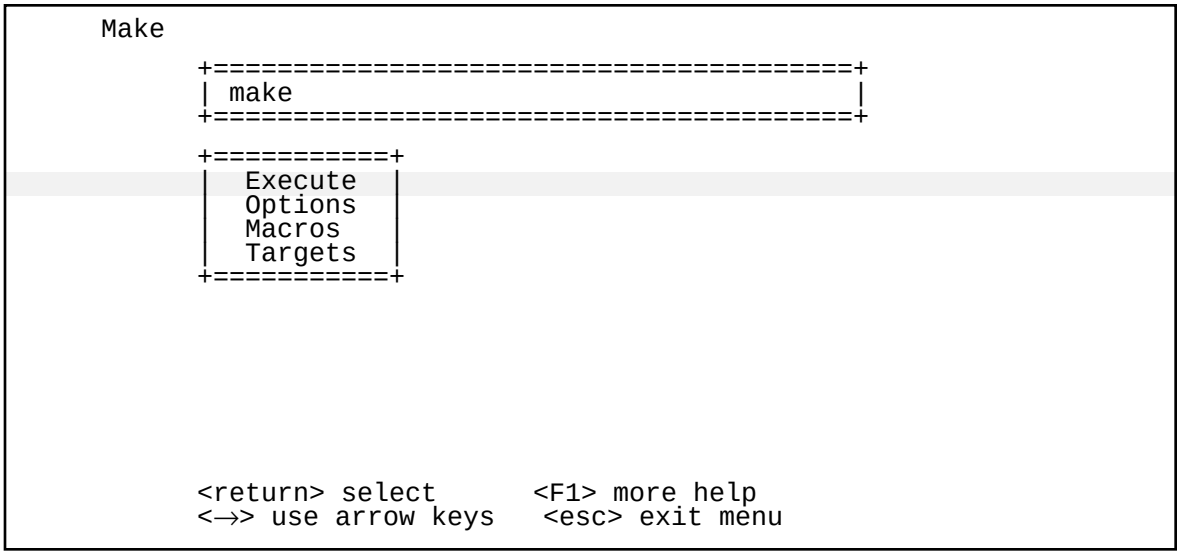
When you are done working with MS-DOS, type **exit**. The MS-DOS prompt will disappear, and the MWS main menu will reappear.

### Using the make programming discipline

MWS gives you quick access to **make**, the Mark Williams programming discipline.

**make** helps you build large, complex programs. It reads a **makefile** that you prepare, and follows the **makefile**'s descriptions to build your program. If you need more information on **make**, see the entry for **make** in the Lexicon, and see the tutorial on **make** that appears in section 5 of this manual. The tutorial also describes in detail how to write a **makefile**.

To invoke **make** through MWS, press the ↓ key until the cursor bar covers **Make**. Type **<return>**. The main menu disappears, and the screen appears as follows:




The cursor bar is over the top entry in the menu, **Execute**. MWS remembers the last command you gave **make**, if any; so, if you press **<return>** MWS will automatically execute the last **make**

command you issued.

If you wish to construct a new **make** command, the first step is to change **make**'s default macros so that **make** will produce the sort of executable program that you want. Note that this step often is not necessary. For more information on **make**'s macros and the files in which they are kept, see the Lexicon entry for **make**; also see the **make** tutorial.

To redefine a macro, press the ↓ key twice, to move the cursor bar to **Macros**; then press **<return>**. MWS will prompt you to enter the new macro; what you type will then be used instead of any macro that has the same name.

Likewise, should you wish to change the default targets that **make** constructs, press the ↓ key until the cursor bar covers **Targets**; then press **<return>**. MWS will prompt you to type the new target or targets that you want to build. For more information on what a target is and how **make** builds one, see the tutorial for **make**, or see the Lexicon entry for **make**.

The next step is to construct the **make** command line. Press the ↑ key to move the cursor bar to **Options**. Press **<return>**. The screen is redrawn to offer you the options for **make**, as follows:

```
Make

      +=======================================+
      | make                                  |
      +=======================================+

+====+
| -d |  (Debug) Verbose output
| -f |  Alternate make filename
| -i |  Ignore error returns and continue
| -n |  Show commands but do not execute
| -p |  Print macro and target descriptions
| -q |  Query target status
| -r |  Do not use built-in rules
| -s |  Silent running
| -t |  Touch all targets to current time
+====+



      <return> select     <backspace> de-select <F1> more help
      <→> use arrow keys  <end> end options     <esc> exit menu
```

To select an option, press the ↓ key until the cursor bar is positioned over it, and then press **<return>**. The option will be written into the command box at the top of the screen. Should you select an option by accident, press **<backspace>**; the option will be erased from the command box at the top of the screen. If you select the **-f** option, MWS will prompt you for the name of the file that you wish to use in place of **makefile**. For more information on **make**'s options, see the entry for **make** in the Lexicon, or see the tutorial on **make** later in this manual.

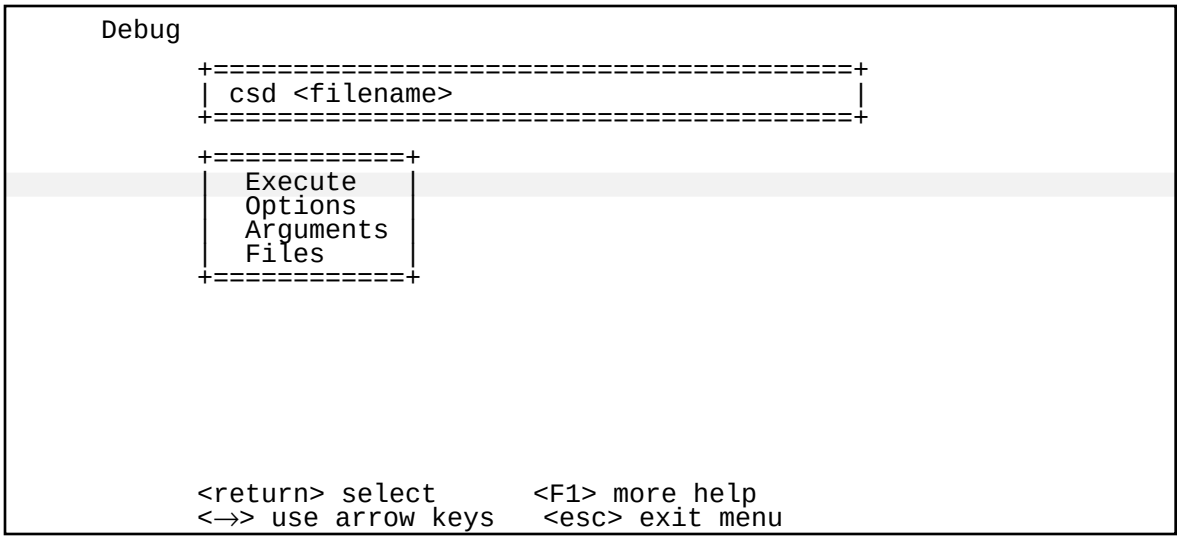
When you have selected all of the options that you want, press **<end>**. You will be returned to the main **Make** menu, and the cursor bar will be positioned over **Execute**. Press **<return>**; MWS will then execute the **make** command that you have built, which appears in the command box at the top of the screen. **make** will look for **makefile**, read it, and begin to build your program. When **make** is finished, you will be returned to the MWS main menu.

### *Using csd, the C source debugger*

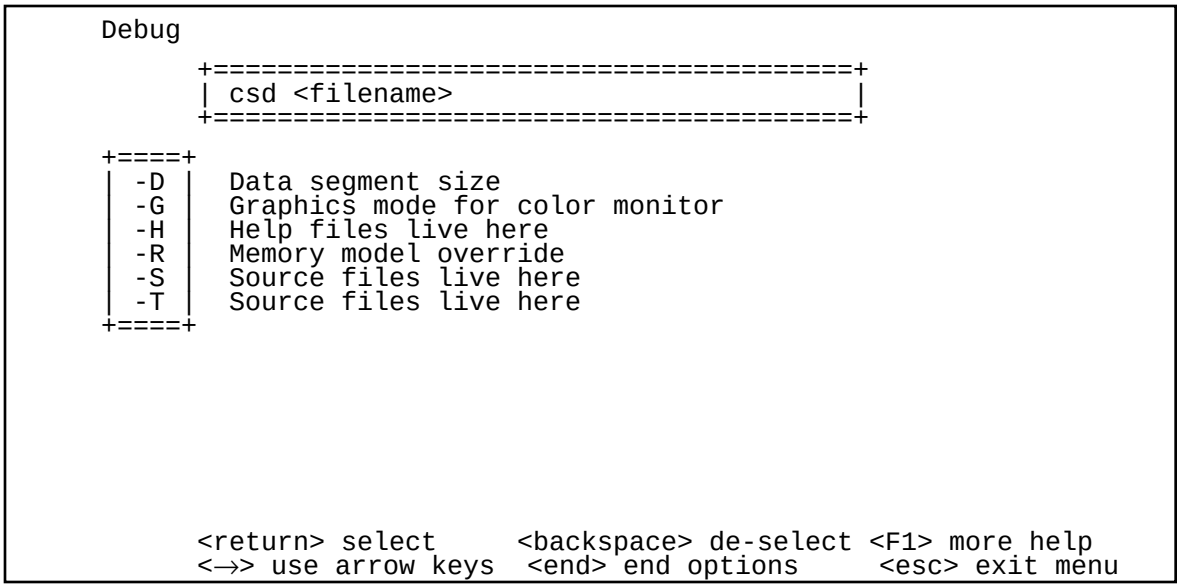
MWS also helps you to invoke **csd**, the Mark Williams C source debugger. **csd** is an invaluable tool to a programmer, for it allows you to debug programs while using your C source code, instead of having to use listings in assembly language. Note that **csd** is not included with **Let's C**, but it is available as a separate product for use with **Let's C**.

If your computer does not have a hard disk, you must insert the disk that came with your copy of **csd** into drive A before you can begin to work with **csd**.

To invoke **cs d**,press the ↓ key until the cursor bar is positioned over **Debug**. Press **<return>**. MWS redraws its screen as follows:




If you wish to construct a new **csd** command, press the ↓ key once, to position the cursor bar over **Options**; then type **<return>**. MWS will redraw its screen to display the available options, as follows:




If you select either **-H** or **-S**, MWS will prompt you to enter the name of the directory where you have stored these files. For more information on **csd**'s options, see your **csd** manual.

As before, if you select an option by accident, press **<backspace>**. This will erase the option from the command box at the top of your screen. When you have selected all of the options that you want, press **<end>**. The list of options will disappear from the screen, and the original **Debug** menu reappear.

To complete your **csd** command, press the ↓ key twice; this moves the cursor bar to **Files**. Press **<return>**. MWS will then draw a box that contains all of the executable files in the current directory. You can select one by moving the cursor bar to it, and then pressing **<return>**.

Remember that to debug a file with **csd**, it must first have been compiled with the **-VCSD** option to the **cc** command line. This writes special debugging information into the executable file.

When you have selected the program you wish to debug and have pressed **<return>**, MWS will write the file name into the command box, and return you to the **Debug** menu; the cursor bar is positioned over **Execute**. Press **<return>** to invoke **csd** and execute the command you have just built. You can then debug your program with **csd** just as it is described in the **csd** manual.

When you have finished your debugging session and have exited from **csd**, you will return to the MWS main menu. MWS will then be ready help you build another command.

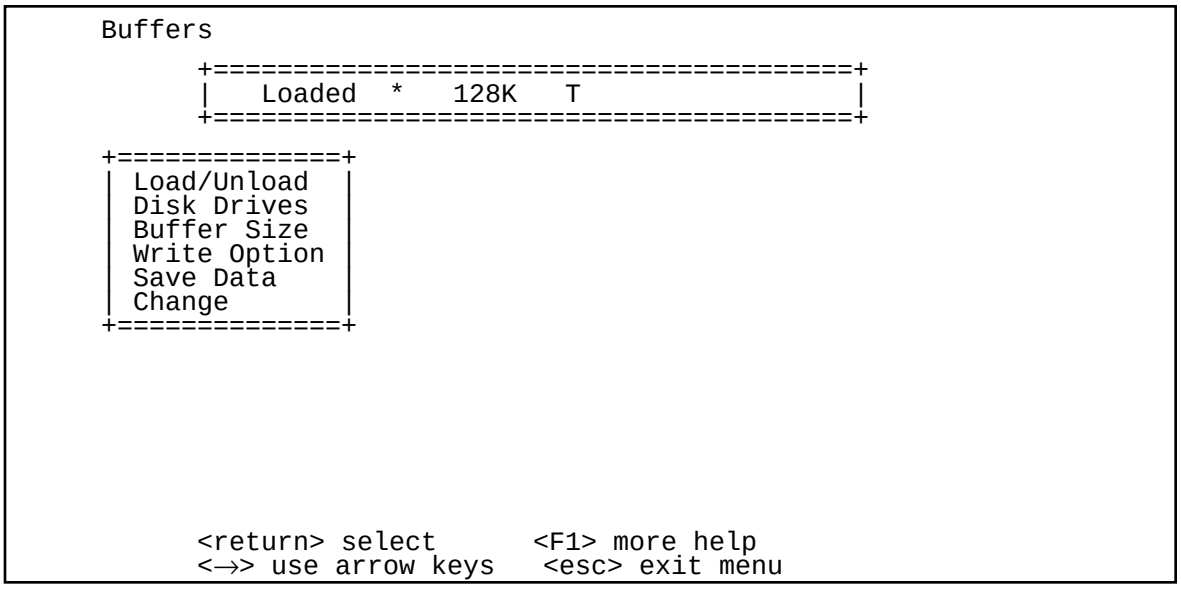
### Resetting the buffers

As noted earlier, MWS not only gives you an easy way to build commands for **Let's C**: it also contains an accelerator that speeds up your software. The accelerator uses a buffering system that reduces the number of times that programs need to read the disk drive. Because reading the disk drive is the slowest part of any program, reducing the number of times the disk must be read increases the speed with which the program operates.

At times, you may need to change how MWS performs its buffering. To do so, use the **Buffers** command on MWS's main menu.

## Let's C

To begin, press the ↓ key until the cursor bar is positioned at the entry **Buffers**. Press **<return>**. MWS will draw the **Buffers** screen, as follows:




Each command is described below.

**Load/Unload**

This tells MWS to load or unload the accelerator. Note that this command is a *toggle*: if the accelerator is unloaded, then pressing **<return>** tells MWS to load it, and vice-versa. Try pressing **<return>**. You will see that the entry in the command box will flip from **Loaded** to **Unloaded**.

Note, too, that this command does *not* take effect immediately. To unload or load the accelerator, you must exit from MWS and then re-enter it.

**Disk Drives**

This command tells MWS which disk drives you want accelerated. The default is '*', which indicates that all drives are accelerated.

To change the settings, press the ↓ key until the cursor bar is positioned over **Disk Drives**. Press **<return>**. MWS will prompt you for the name of a disk drive. Type **A**, to indicate that you want to accelerate disk drive A. Note that the asterisk in the command box has been replaced with **A:**. The prompt remains on the screen: if you wish, you can name any number of disk drives. To end this session, press **<return>** without typing anything. MWS will now accelerate only disk drive **A:**.

Now, re-invoke this command by moving the cursor bar to **Disk Drives** and pressing **<return>**. Type '*', and then press **<return>** twice. MWS has now resumed accelerating all disk drives.

**Buffer Size**

The accelerator reserves a portion of memory to buffer what it reads from your disk drives. By default, the size of its buffer is 128 kilobytes, as shown in the command box. If your system has limited amounts of memory, you may wish to decrease this amount. On the other hand, if your system has memory to spare you may wish to increase the size of the buffer: the larger the buffer is, the more your software will be accelerated.

To change the size of the buffer, press the ↓ key until the cursor bar is positioned over **Buffer Size**, and then press **<return>**. MWS will prompt you for the new buffer size. If you change your mind and decide to leave the buffer unchanged, simply press **<return>** without entering anything.

Note that this command, like the **Load/Unload**, does *not* take effect immediately. You must leave MWS and then re-enter it before you can begin working with a different sized buffer.

**Write Option**

The MWS accelerator not only speeds up the rate with which your programs read the disk: it also speeds up the rate at which they *write* data to the disk.

The accelerator offers you three ways to save your data to the disk: **Memory**, **Timed Save**, and **Disk**. The **Memory** option stores all data in memory. No data are written to the disk until you choose to save them with the **Save Data** command, which will be described in a moment. The **Disk** option writes all data directly to disk, and does not buffer data at all. Note that the **Memory** option is faster than the **Disk**, but not as safe, because an accident could cause you to lose the data stored in the buffer.

The **Timed Save** option combines features of the **Memory** and **Disk** options. Data are stored in memory, but they are automatically written to disk every five minutes. Thus, you have the speed of in-memory storage, plus the safety of saving data to disk. This is the default option.

To change the write option, press the ↓ key until the cursor bar is over **Write Option**, and then press **<return>**. A menu will appear that displays the three write options. Move the cursor bar to the one you want and then press **<return>**. The option you select will be shown in the command box, and you will be returned to the **Buffers** menu.

**Save Data**

This command writes to disk all data that MWS has stored in its buffer. To use it, simply press the ↓ key until the cursor bar is positioned over **Save Data**, and then press **<return>**. This command has no options: it will simply write the data, and return you to the **Buffers** menu.

You should use this command to save your data before you turn off your machine or before you leave MWS, should you be using the **Memory** option.

**Change Disk**

The last command, **Change Disk**, *must* be used before you change the floppy disk in a disk drive that is being accelerated. This command writes to disk all of the data that MWS has stored in its buffer, and then empties the buffer to make it ready for the new disk.

If you do not use this command before you change a floppy disk, you could lose data. This bears repeating: *You must use this command before you change a floppy disk in a drive that is being accelerated, or you could lose data.*

To use this command, simply press the ↓ key until the cursor bar is positioned over **Change Disk**. Press **<return>**. Any saved data will be written out, the buffer will be emptied, and MWS will be ready to accept a new floppy disk.

To return to the main MWS menu, press **<esc>**.

As you can see, MWS's accelerator is easy to use. With a few simple commands, you can alter its settings to suit your preferences. MWS will remember these settings, and use them automatically in the future.

## Let's C

### Where to go from here

The following bibliography lists reference books on C and on the i8086 processor. This list includes both primers and references for advanced programmers. This list is not exhaustive, but you will find it helpful should you need detailed information on a topic.

Section 2, *C for Beginners*, introduces the C language to users who are new to C. If you are experienced with C you may wish to skip this section, but if you are a novice at C programming you may find it helpful.

After *C for Beginners* is a section on *Advanced Compiling*. This discusses many of the compiler's options that were just mentioned here. It also discusses some technical issues on the i8086 microprocessor that both experienced programmers and novices will find helpful.

Finally, if you need more information on any command, library function, C keyword, or technical term, check the Lexicon.