Introduction

Congratulations on choosing **csd**, the Mark Williams C source debugger. **csd** can speed the development of any C program and make the C language more accessible to the novice programmer and expert alike.

C is a powerful language that combines the flexibility, speed, and compactness of assembly language with higher level language features such as data structures, control structures, and functions. C programs are highly portable and are not tied to a specific machine architecture. Compilers like **Let's C** brought C programming to i8086-based microprocessor systems. Programs that once required assembly language could now be written in C. However, the lack of a true source-level debugger meant that C programs still had to be debugged at machine level.

Now, with **csd**, you can debug your C programs in the same language in which you wrote them. **csd** displays your source code, evaluated expressions, all traced expressions, and program output, each in its own window. Program locations are referenced by their positions in your source file and variables are referenced by name, relieving you of the chore of looking up and calculating numeric machine addresses. If you are a new programmer, **csd** can shorten the time you need to become productive because you no longer need to learn assembly language to debug your programs.

Debugging is the most time-consuming part of program development. **csd** helps make this chore manageable. It is indispensable as a learning tool for newcomers to C, and a time saver for experienced programmers.

Features

csd includes the following features:

- Four different windows let you examine your source code in detail, see your actual program output displayed, evaluate C expressions, and keep a log of traced statements and expressions.
- **csd** lets you enter and evaluate new C expressions, without having to recompile your program.
- With **csd** you can *trace* any executable C statement, stopping execution immediately before the marked line of source.
- You can also trace expressions in the evaluation window. Execution will stop each time an expression marked as traced changes value.
- **csd** gives you a complete list of traced statements and expressions. Each time a program stops at a traced expression, a copy of that line is written to the *history window*, maintaining a complete list of traced statements and expressions.
- On-line help screens make **csd** easy to use, even without a manual.
- You can debug both large and small model C programs with **csd**.

csd does for debugging what high-level languages do for programming: it makes your work easier, saves time, and increases your productivity.

What is csd?

2 Introduction

The task of developing a program can be broken down into several stages, each of which has its own problems.

The first stage is *planning*. At this stage you identify a problem that needs solving, and determine how it can be solved on a computer.

The second stage is *design*. Here, you break the problem into discrete tasks, or *functions*, and sketch out how a program will link the tasks into a whole.

The third stage is *coding*. You translate each function into C code, then compile and link it into an executable program.

The fourth stage is *debugging*. This is the most tedious part of programming. At this stage you methodically test the program, find out where it does not work properly, and fix it. Bugs can result from mistakes at any stage of writing a program: the planning may be faulty, the design may incorrectly interpret the plan, and the coding may have mistakes in it.

Debugging code is often the most difficult and time-consuming part of programming. It is easy to fix a program; the hard part is finding just what is wrong. All experienced programmers can tell "horror stories" about debugging programs that simply do not work. You can study the output to find where the problem might lie. Then you embed **printf** statements within the code to print out the values of certain variables, then recompile and rerun the program; if this does not help, then you embed more **printf** statements elsewhere, recompile, and try again. This continues through perhaps dozens of recompilations until you stumble across the problem. Throughout this process, you find yourself wishing you had a tool that would let you watch the program run so you could stop execution at selected points in the program and examine the values of variables to discover just where the program is going wrong.

csd is such a tool. It lets you peer into your program while it is running. You can run your program from beginning to end, from beginning to any point in the code, or from any point in the code to any other, either all at once or one step at a time. At any point in execution, you can see what the program has output; you can look into memory and see the value of any variable; you can explore the stack and see what functions are calling other functions; and you can enter new C expressions and evaluate them immediately. **csd** does all of this interactively, using windows and keystrokes. You no longer need to alter your program and recompile in order to see how your program is working.

How does csd work?

In essence, **csd** performs two tasks: (1) it starts and stops the execution of a program, and (2) it lets you examine and alter the contents of memory.

Controlling execution

When you invoke **csd**, it loads your program and runs a portion of it. It then saves your *program window* (that is, the screen on which the program writes its output), and displays your source code in the *source window*, with a cursor positioned at the top of the first source file.

You can set a *breakpoint* on any line of source simply by moving the cursor to that line and pressing a key; a breakpoint is a point at which **csd** automatically stops executing the program. A line that bears a breakpoint is said to be *traced*. When you run the program, **csd** starts executing the program from where it last stopped (or from the beginning of the program) to immediately before execution of the next traced line it encounters.

When **csd** stops execution, it saves the program window, retaining what your program has output up to this point. It then redraws the source window, with the cursor positioned at the line of source code at which execution stopped.

You can run your program any number of times while using **csd**.

Referencing data

csd also has an *evaluation window*. There, you may type any C expression that would be valid at your current location in the source window. If the expression is also valid at your current execution position (which may or may not be the same as your current source window position), it is evaluated immediately and its value is displayed. The expression stays in the evaluation window until you delete it.

Since expressions may involve assignment operators (++, --, =, +=, etc.) you can alter your data. The program itself is never altered. Because data alteration may affect the future operation of your program while debugging, this is one way to influence execution. Whenever your program is stopped, every expression in the evaluation window that is valid at the current stopping position is evaluated and its value is displayed.

You can also trace any expression in the evaluation window. **csd** can be instructed to run your program until the value of one of the traced expressions changes.

How to use this manual

This manual introduces you to **csd**. It assumes that you are familiar with the C programming language. ", as well as with DOS and its commands."

This manual also contains everything you need to use **csd** to its fullest advantage. It is organized into the following sections:

1. *Introduction*.
2. *Becoming familiar with* **csd**. This tutorial walks you through the sample C program,**infl**, to demonstrate **csd**'s basic functions.
3. *Advanced features*. Here, the tutorial walks through a more complex program, **factor**, to demonstrate advanced **csd** features. The use of both tracepoints and the evaluation window are two of the features described in greater detail.
4. *A sample debugging session*. This section uses the program **inflbug** to demonstrate how you can use **csd** to track down some of the more common C programming bugs.
5. *Questions and answers*. This section presents a number of questions that new users commonly ask about **csd**. If you have a question about **csd** or its use, look here first.
6. *Commands reference*. This section summarizes all of **csd**'s commands.
7. *Help screens*. This section reproduces all of **csd**'s help screens. It can also be used as a quick reference.
8. *Error messages*. This last section presents all of **csd**'s error messages, discusses what each one means, and gives hints on how to address the problem.

Conventions used in this manual

This manual represents the cursor by a block character '▮'. On your screen, of course, it will be a flashing underscore or block.

Text that is highlighted on your screen is represented in this manual by shaded print.

Sections 2 through 4 of this manual are tutorials on using **csd**. They also reproduce a number of sample screens. As you work through the tutorial, the displays on your computer may differ slightly in other ways from those shown in this manual. For example, you may be logged into a different drive, so it may say **A>** rather than **C>**. These differences should be minor, and will not affect how

csd operates on your computer.

User registration and reaction report

Before you go any further, please fill out the User Registration Card that came with your copy of **csd**. Returning this card will make you eligible for direct telephone support from the Mark Williams technical staff. Also, if you have comments or reactions to the **csd** software or documentation, please fill out and mail the User Reaction Report included at the end of the manual. We especially wish to know if you found errors in this manual. Mark Williams Company needs your comments to continue to improve **csd**.

Installing csd

The following files are on the **csd** distribution disk:

CSDXL.OBJ	CSDSELEC.HLP
CSDXS.OBJ	CSDTRACE.HLP
CSD.EXE	ATOD.C
LCSD.EXE	FACTOR.C
CSDEVAL.HLP	FACTOR.EXE
CSDEDIT.HLP	INFL.C
CSDFIND.HLP	INFL.EXE
CSDHELP.HLP	CC4.EXE
CSDRUN.HLP	INSTALL.DAT
INSTALL.EXE	INFLBUG.C

Before you begin to use **csd**, be sure to make a backup copy of the distribution disk. *Never work with the distribution disk: always work with a copy.* When you have made your backup copy, put the distribution disk away in a safe place.

Installing **csd** involves copying files from the distribution disk onto either a hard disk or a floppy disk. To use **csd** with a two-floppy system, simply copy the distribution disk to a formatted disk.

To copy **csd** to a hard disk, use **csd**'s **install** utility, which does the copying for you. By running **install** and answering a few simple questions, you can build a working copy of **csd** on your hard-disk system in a few minutes.

Installing csd onto a hard disk

To begin, log onto drive C on your system. On nearly all computers, this is the hard disk. Then insert the **csd** distribution disk 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 wish to install all 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 executable programs(default: "\bin")?
Where do you want libraries (default: "\lib")?
Where do you want sample programs (default: "\sample")?
```

After each question, type **<return>**, which accepts the default setting. Later you may wish to re-install **csd** into other directories of your own choice, but at present it is best to use the default settings.

install will now begin to copy the files from the distribution disk onto your hard disk.

csd C source debugger

That's all there is to it. **csd** is now installed on your hard disk.

How to run csd

To debug a program with **csd**, you must first compile it with the **Let's C** compiler using the **-VCSD** option to the **cc** command line. For example, the sample program **infl**, which is included on your **csd** distribution disk, was compiled using the command line:

```
cc -f -VCSD infl.c
```

If you are using the Mark Williams Shell, MWS, be sure to select the **CSD** variant option when you compile. For complete information on compile command line options and MWS, see the **Let's C** manual.

Invoking csd through the Mark Williams Shell

It is recommended that you use **csd** through MWS, the Mark Williams Shell. The shell contains a disk accelerator that will speed up **csd** noticeably. In addition, MWS gives you two ways to enter **csd** commands: either through its graphics interface, which helps you build command lines, or directly through **command.com**.

To use **csd** through the MWS graphics interface, do the following: First, invoke MWS by inserting the **shell** disk from **Let's C** into drive A (if you have a two-floppy disk system) and typing MWS. Remove the **shell** disk from drive A and insert your backup copy of the **csd** disk.

Now, press the down-arrow key <↓> until the reverse-video band, called the *cursor bar*, covers the **Debug** on the main menu. Press **<return>**. MWS will draw a new screen for you; this screen gives you access to all of **csd**'s options and features.

Press the <↓> key until the cursor bar is at **Files**; press **<return>**. A new box will open on the screen; this box displays all of the executable files that are available in the current directory. Press the <↓> key again until the cursor bar covers the entry **infl.exe**; press **<return>**. As you can see, the name **infl.exe** has been written into the command box at the top of the screen and the cursor bar has returned to the entry **Execute** on the main **Debug** menu. Now, press **<return>** again. This executes **csd** with the program **infl.exe**

If you prefer not to use the graphics interface, do the following. First, invoke MWS as described above. Then remove the **shell** disk and insert the **csd** disk into drive A. When MWS's main menu appears, press the <↓> key until the cursor bar covers the entry **!DOS Escape**. Press **<return>**. The screen will clear, and your normal DOS prompt will appear. Now you can type commands directly into DOS; however, MWS will still be working in the background to accelerate your programs. To return to MWS, simply type **exit**, and the MWS main menu will reappear.

To invoke **csd** now, simply type

```
csd infl
```

at the DOS prompt.

Using csd

Once you have installed **csd**, try running the program on the sample program **infl**. **infl** is already compiled with the appropriate command line options, and is ready to debug.

To invoke **csd**, type the following command from the MS-DOS prompt:

```
csd infl
```

csd will load your program and display the beginning of the source code on your screen. Your screen should appear as follows:

```
#include <stdio.h>
main()
{
    int i;                          /* count ten years */
    float w1, w2, w3; /* three inflated quantities */
    char *msg = " %2d\t%f %f %f\n"; /* printf string */

    i = 0;
    w1 = 1.0;
    w2 = 1.0;
    w3 = 1.0;
    for (i = 1; i <= 10; i++) {  /* apply inflation */
        w1 *= 1.07;
        w2 *= 1.08;
        w3 *= 1.10;
        printf (msg, i, w1, w2, w3);
```

infl.c

To make **csd** run the program, type the *run* key, **<F4>**, followed by the *end* key, <End> . The *program window*, where your program writes its output, will appear briefly as **infl.exe** executes; then **csd** displays the source window with your program's source code.

Now, press the *program* key, **<F7>**. **csd** redisplays the program window so you can study the output of your program. You will see the following screen:

```
C>csd infl
C Source Language Debugger Version 1.1
Copyright (c) 1984-1988 by Mark Williams Company, Lake Bluff, IL
Reading source...
Loading...
  1   1.070000 1.080000 1.100000
  2   1.144900 1.166400 1.210000
  3   1.225043 1.259712 1.331000
  4   1.310796 1.360489 1.464100
  5   1.402551 1.469328 1.610510
  6   1.500730 1.586874 1.771560
  7   1.605781 1.713824 1.948716
  8   1.718186 1.850930 2.143588
  9   1.838458 1.999004 2.357946
 10   1.967150 2.158924 2.593741
```

Now, type the *source* key, **<F8>.** The screen will again display the source code.

It is this easy to use **csd**: one command invokes the debugger and displays your source code, two keystrokes execute your program, one keystroke displays the results for you and another returns you to the source code to continue debugging.

Now, type **<Shift-F1>**, the *exit* key, to exit **csd**.

Where to go from here

If you are a newcomer to C programming, or have never used a source level debugger, you should work through sections 2 through 4 to learn how to use **csd**. Experienced programmers will want to go directly to sections 6 through 8. The *Commands reference* section, along with **csd**'s on-line help screens, should be all you need to get started with **csd**.