A sample debugging session

This section of the manual contains exercises that let you use **csd** on a program with bugs. An error-filled version of the program **infl** named **inflbug** will be used to demonstrate some common C language bugs and how to find them with **csd**. **inflbug.c** is included on your **csd** distribution disk.

To begin, compile **inflbug.c** using the following command line:

```
cc -f -A -VCSD inflbug.c
```

Note the **-VCSD** option, which tells the compiler to include the information needed to run your program under **csd**. Note also the **-A** option. This option automatically calls the MicroEMACS screen editor when the compiler encounters an error during compilation. The errors are displayed in one window, and the source code file in the other, with the cursor set to the line number indicated by the first error message. Typing **<ctrl-X>>** moves to the next error, and **<ctrl-X><** moves to the previous error. To recompile, close the edited file with **<ctrl-Z>**. Compilation will continue either until the program compiles without an error, or until you exit the editor by typing **<ctrl-U>** followed by **<ctrl-X><ctrl-C>**.

inflbug.c will compile without an error message.

Where to start

As you know from section 2, the program **infl** computes three different rates of inflation over a span of ten years. **inflbug** is supposed to do exactly that, but fails. To set up this example, the errors in **inflbug.c** have been created so that the program will compile but not execute.

Type

```
inflbug
```

at the MS-DOS prompt. **inflbug** runs, but instead of producing a chart of inflated values as the program **infl** did in section 2, it does nothing. If you had written this program, you would have planned for the program to compute the inflated values ten times, and print the results in a chart to your screen. Since the program compiled without an error message, you can now use **csd** on the program to find just where the program fails.

Type

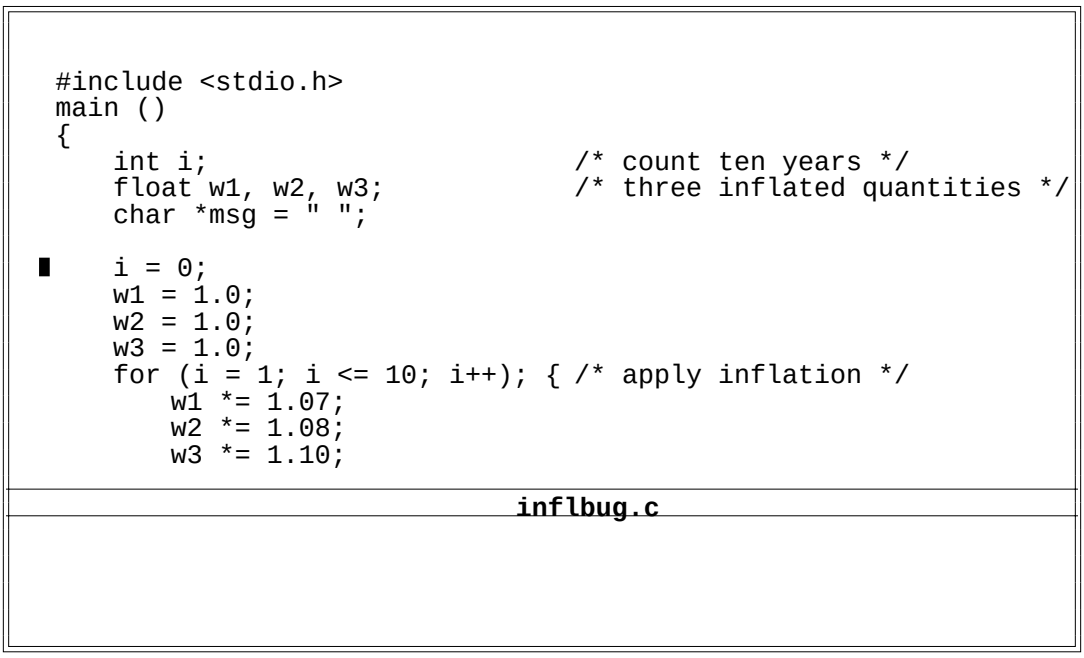
```
csd inflbug
```

The source window will appear, with the beginning of the program displayed in the source window. Your screen will appear as follows:

```
#include <stdio.h>
 main ()
 {
     int i;                          /* count ten years */
     float w1, w2, w3;  /* three inflated quantities */
     char *msg = " ";

     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;
```

inflbug.c

To begin debugging, single-step through the program. Type **<F4>** followed by **<return>** to execute one line of **inflbug**. The screen will look as follows:



Notice that the cursor is positioned at the first executable line of code. Continue single-stepping through the program by typing **<F4>** followed by **<return>**. After each step, type **<F7>** to change from the source window to the program window after executing each line to see what output, if any, the program has produced. Return to the source window by typing **<F8>**. As the program executes each line, the cursor will move to that line of code.

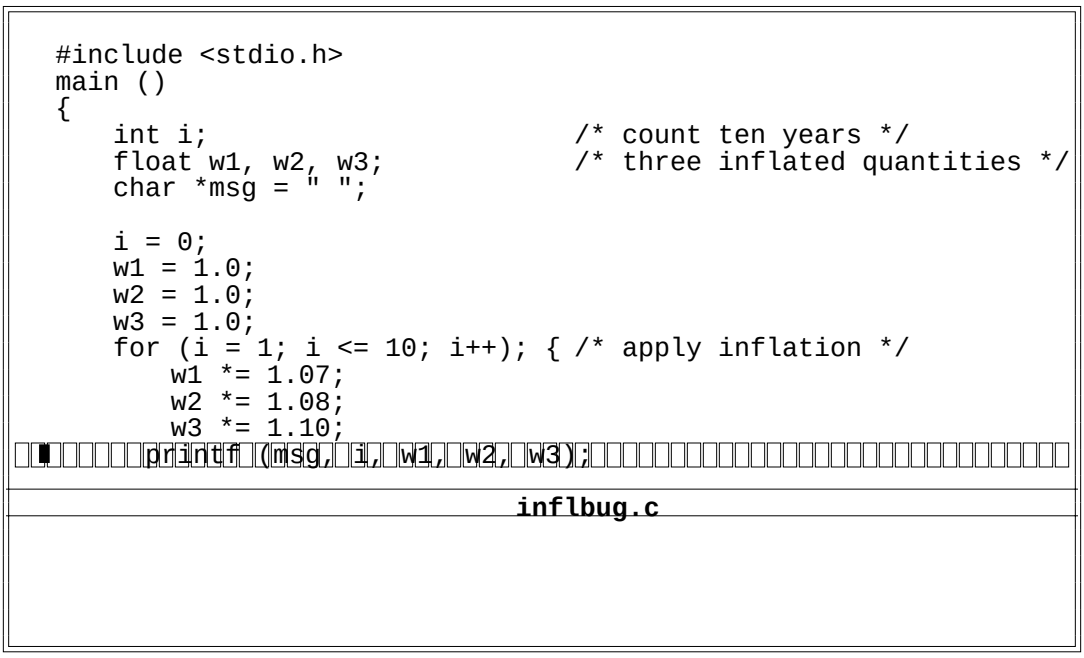
As you watch **inflbug** execute each line, you will notice that you must type **<F4>**, then return ten times after you reach the 'for' loop before the cursor moves on to the statements following the **for** statement. It then proceeds through the rest of the loop, stopping once before each statement. Then the program exits, leaving the cursor positioned at the upper left corner of the source window.

Switch to the program window by typing **<F7>**. The program window looks like this:

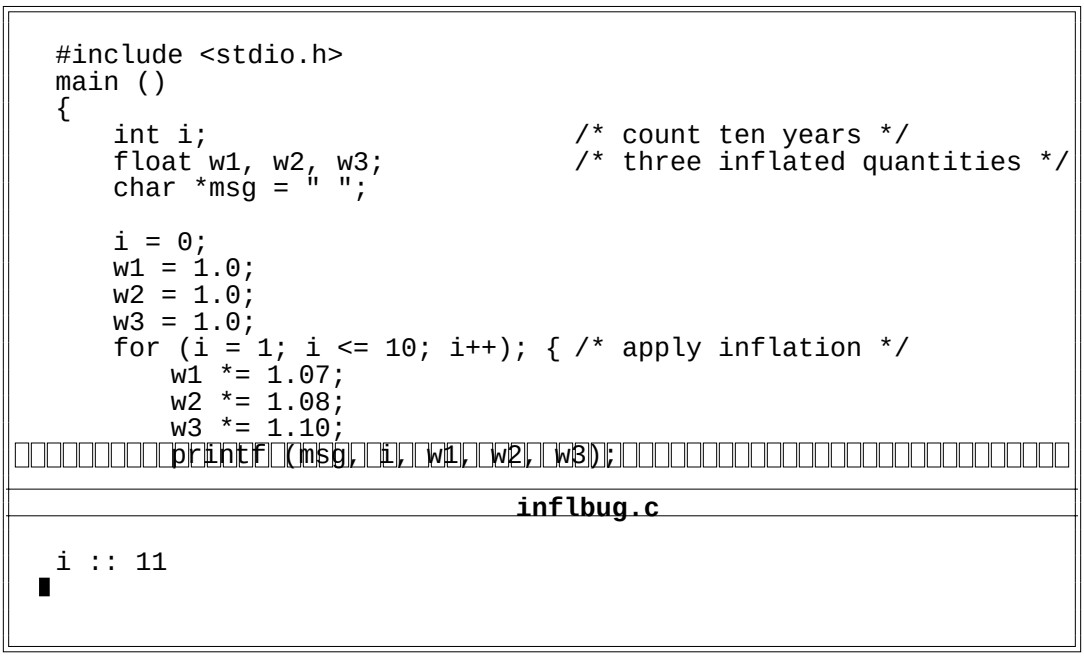
```
C>inflbug


C> csd inflbug
C Source Language Debugger Version 1.1
Copyright (c) 1984-1988 by Mark Williams Co., Lake Bluff, IL
Reading source...
Loading...
```

The program should execute the entire **for** loop; instead, it proceeds only with the **for** *statement*, incrementing the variable **i**. It does not perform the operations in the rest of the loop until after it has executed the **for** statement ten times. It seems as if the **for** statement is isolated from the rest of the loop. Type **<F8>** to return to the source window.

To isolate the problem further, set a tracepoint on the **printf** statement. Use the **<↓>** key or **<crtl-N>** to position the cursor at the **printf** statement, then type **<F3>**. The **printf** function is the last executable statement of the **for** loop. By stopping program execution here, you can see the value of the program's variables after one pass through the loop. Now, run the program to the traced point by typing **<F4>** then **<F3>.** This causes the debugger to execute the program for what should be one pass through the **for** loop. Your screen looks like this:



Check the value of **i**: move to the evaluation window by typing **<F9>**. Type **i** and **<return>**. The line in the evaluation window shows



i is a variable that is incremented to set a test-condition for execution of the **for** loop. Each time the loop is executed, the value of **i** increases by one. Since the variable has a value of 11, you know that the **for** statement has been executed ten times before it reached the traced **printf** at the end of the loop. Now, check the value of **w1** to see if it has been affected by the statement

```
w1*=1.07;
```

The operator '*=' should multiply the variable **w1** (which has a declared value of one), by 1.07, then assign the product as the new value for **w1**. Type

```
w1
```

in the evaluation window, then **<return>**. The results will be:

```
#include <stdio.h>
main ()
{
    int i;                          /* count ten years */
    float w1, w2, w3;               /* three inflated quantities */
    char *msg = " ";

    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);
```

inflbug.c

```
i :: 11
w1 :: 1.07
```

This means that the variable **w1** has been multiplied by the inflation rate once, even though the **for** statement has made ten iterations. From this information, you can conclude that even though the **for** loop has been incrementing, the steps within the loop have not been executed. Somehow the test-condition statement is separated from the rest of the instructions in the loop.

Take a closer look at the **for** statement:

```
for (i = 1; i <=10; i++);
```

The loop test says that while **i** is less than or equal to 10, increment it by one; but it ends there, and does not continue with the rest of the loop because of the semi-colon ';' at the end of the line. So, **csd** runs the **for** statement instructions the designated number of times and goes on to the statements which apply the inflation rate. It proceeds to the **printf** statement, then exits.

Editing your program

Once you have tracked down the bug in this program, you need to edit out the extra semi-colon and recompile the program. Exit from **csd** by typing **<Shift-F1>**. Use MicroEMACS to edit your program. The source code for **inflbug** is in the file **inflbug.c**. To open the file, type

```
me inflbug.c
```

The screen clears, and in a moment the source code appears. Type **<ctrl-N>** to move the cursor to the **for** statement, then type **<ctrl-F>** until the cursor is positioned just to the right of the extra semi-colon at the end of the **for** statement. Type the delete key **<Del>** to remove the semi-colon. Now save the text and exit MicroEMACS by typing **<ctrl-Z>**. For more information on how to use the editor, see the MicroEMACS tutorial in the manual for **Let's C**.

Recompiling your program

If you change your source file, as you have in this tutorial, you must recompile the program with the **-VCSD** option on the compile command line. After you have returned to the prompt, type this compile command line:

```
cc -f -A -VCSD inflbug.c
```

As discussed at the beginning of this section, the **-A** option on the command line tells the compiler program to invoke MicroEMACS whenever it encounters an error message. If you have made any errors while editing the source code text, you will be returned to MicroEMACS to correct the program. Exiting from MicroEMACS automatically recompiles the program. This compile-edit-compile cycle will continue until your program is error free, or until you exit by typing **<ctrl-U>**, followed by **<ctrl-X><ctrl-C>**.

When you have finished compiling, try running your program again. At the system prompt, type

```
inflbug
```

Another bug

The program still produces no output. Invoke **csd** on the program again. Type

```
csd inflbug
```

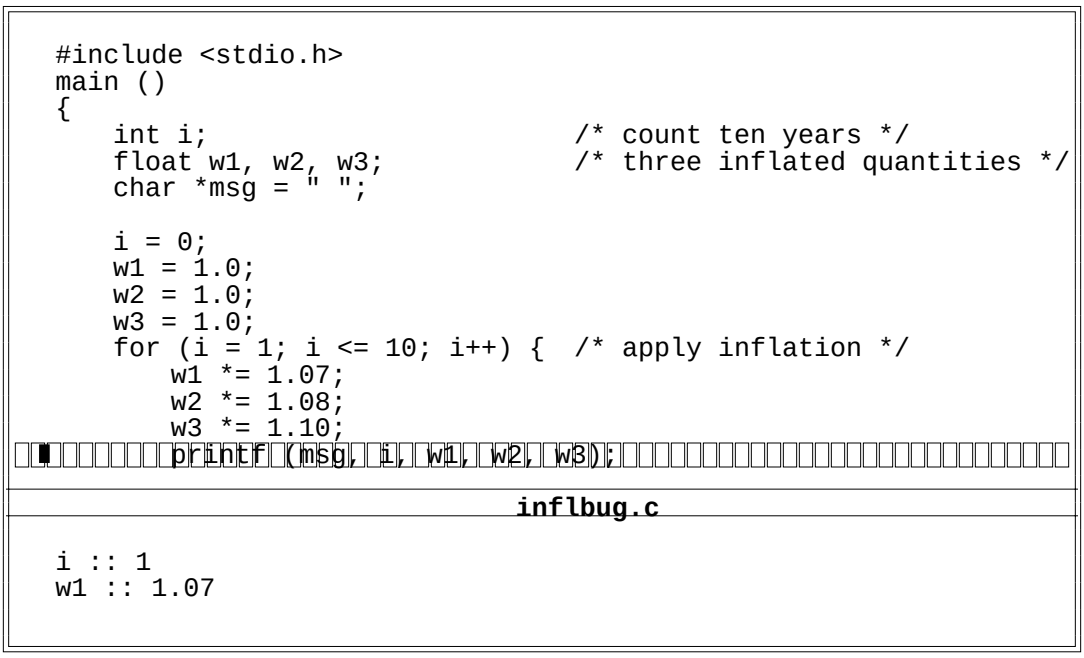
and **<return>**.

Using the cursor movement keys, position the cursor at the **printf** call, and type **<F3>** to set a trace point.

Press **<F4>** followed by **<F3>**, and the program will execute to the traced line. Now, enter the evaluation window by typing **<F9>**. Type

```
i
w1
```

csd will return the following values in the evaluation window:



Now run the program through again by typing **<F4>** followed by **<F3>**. The program will run once through the loop. Type **<F4>** then **<F3>** repeatedly, and watch the value of the variables you typed in the evaluation window change. When you have run **inflbug** to its end, your screen will appear as follows:

```
#include <stdio.h>
main ()
{
    int i;                          /* count ten years */
    float w1, w2, w3;               /* three inflated quantities */
    char *msg = " ";

    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);
```

inflbug.c

```
i :: 10
w1 :: 1.967150
```

By checking the value of **i** and **w1** with each iteration of the **for** loop, you can see that the program is operating upon the variables and executing the loop properly. The variable **i** has been incremented for the test-condition of the **for** loop, and the variable **w1** has been operated on with the appropriate results. However, none of the computed values are being printed to the program window; therefore, the problem must be in the **printf** call.

Since **printf** is a format conversion function, it should contain instructions to format a line for printing the variables. This format has conversion specifications which take the value of **i** as the first number on the output line (the line number). The inflated variables are the rest of its arguments. **msg** is the pointer to a string which should contain the conversion specifications. Evaluate the string **msg** using the string cast **(str)**. Move the cursor so that it is inside **main**. Return to the evaluation window and type:

```
(str)msg
```

in the evaluation window. The result is:

```
(str)msg :: ""
```

The pointer has not been initialized to hold the **printf** conversion specifications, so it picks up a random value in memory. The pointer is intended to contain the address of a string, but no address has been assigned to it. The string therefore contains whatever was at that location on the stack when the program was run. **msg** does not point to a valid format, and **printf** does nothing.

Calling functions in the evaluation window

Calling functions from the evaluation window is a powerful debugging tool. In this sample debugging session, you can use this tool to test a modified **printf** call.

Type the following line in the evaluation window:

```
printf("  %2d         %f %f %f\n", i, w1, w2, w3)
```

Now, restart the program by typing **<F4>** followed by <Home> . Run the program to its end: type **<F4>** then <End> . Check the program window; you will see the effect of calling this routine in the

evaluation window. All of the **for** loop operations have been displayed in the format called for in the new **printf** statement; in addition, the **printf** statement already in the program prints its output with each iteration of the loop.

By calling a new **printf** routine, you can see that while your variables are being incremented as you planned, there is something wrong with the **printf** statement: it does not have a format string.

Exit, edit, and recompile

Exit **csd** by typing **<Shift-F1>**. Then invoke MicroEMACS on the file **inflbug.c**. Type

```
me inflbug.c
```

With **<ctrl-N>**, move the cursor to the line

```
char *msg = " ";
```

Type **<ctrl-K>** to remove that line. The cursor will be positioned at the beginning of the line. Type a **<Tab>**, then:

```
char *msg = " %2d       %f %f %f\n";
```

Now save your changes and exit MicroEMACS by typing **<ctrl-Z>**. Recompile as before, using the following command line:

```
cc -f -A -VCSD inflbug.c
```

If you run the program **inflbug** by typing

```
inflbug
```

you should get the results you expected, as follows:

```
C>inflbug
  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
 C>
```

Where to go from here

This section used the program **inflbug** to demonstrate the use of **csd** on a program with bugs. Special attention was paid to the use of the **-VCSD** option in the command line, as well as the **-A** option to call MicroEMACS from the compile command line. Practical examples of single-stepping, setting tracepoints, and calling functions in the evaluation window were shown.

Sections 6 through 9 will give detailed references for **csd** commands, as well as copies of the **csd** help screens, an explanation of error messages, and some commonly asked questions. You have seen the use of **csd** on some programs; with the *Commands reference* section and the **csd** on-line help screens as guides, you will be able to use **csd** to debug your own C programs.