CSCU9V4 Pract. 5:“Oh dear.” (Debugging)

Introduction

Debugging is the process of finding compile time and run time errors in the code. Compile time errors occur due to misuse of C language constructs. Most of the compile time errors can be overcome with careful attention to language syntax and use. As programs grow large with multiple files, IDEs do a good job of pointing to locations of errors in the code.

At the command line, however, one tip would be to compile source files separately. For example, a program made of two files `main.c` and a `mylib.c` can be compiled by,

```
% gcc –ansi –Wall –pedantic mylib.c main.c –o prog
```

Best to compile files individually until they are bug free. This is done using the **–c** flag (compile only),

```
% gcc –c –ansi –Wall –pedantic main.c
% gcc –c –ansi –Wall –pedantic lib.c
```

Then link with,

```
% gcc lib.o main.o –o prog //add .exe if in Windows.
```

Note that IDEs can quite happily compile individual files!

Runtime Errors with Debugging Statements

Consider the following code:

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

int sum(int x, int y, int *z) {
  char c = 4;

  return (x + y + *z);
}

int main(int argc, char *argv[]) {
  int i, j, k, *z;
  int result;
  *z = k;

  if (argc != 4) {
     printf("Please specify three numbers as parameters.\n");
     exit(1);
  }

  i = atoi(argv[1]);
  j = atoi(argv[2]);
  k = atoi(argv[3]);
  k *= 2;

  result = sum(i, j, z) + sum(i, j, &k);

  printf("%d\n", result);
```

```
  return 0;
}
```

NOTE: To add run-time parameters in Netbeans, right-click the project, select Properties→Run and append parameters to 'Run Command'. Here, we might append '5 7 9' (no quotes).

The program has problems that can be found with closer inspection. However, debugging statements are often useful to narrow the search field. Try and run the program above; at best there will appear an exit code of 1, at worst a bus error.

This begs the question: How do we find run-time errors?

Task 1. Modify the program to include debugging statements. In `main()`,

```
fprintf(stderr, "Number of parameters = %d\n", argc);
```

And in appropriate locations in `sum()`,

```
fprintf(stderr, "x=%d\n", x);
fprintf(stderr, "y=%d\n", y);
fprintf(stderr, "z=%d\n", z);
fprintf(stderr, "*z=%d\n", *z);
```

Question: Why is `fprintf()` as above used instead of `printf()`?

Switching Debug Messages On/Off

A quick shortcut to turn debug messages on and off:

1. Beneath the `#includes`, type **`#define DEBUG`**
2. Before each debugging statement, type **`#ifdef DEBUG`**
3. Make sure to write **`#endif`** after each (set of) debugging statement(s).

So, as an example:

```
…

#define DEBUG
…
#ifdef DEBUG
  fprintf(stderr, "x=%d\n", x);
#endif
…
```

As an aside, symbols can be defined at the command line (eg. as a run-time argument) using –D, in this case "`-D DEBUG`".

Using the Debugger

This method is independent from printing debugging statements. **Before continuing, comment out the '`#define DEBUG`' line.**

Debuggers are used to inspect program state during execution. This requires additional information. First, make sure to use the –g option (this is a compile-time option). In Netbeans:

- Right-click your project **Properties** → **Build** → **C Compiler** →**Additional Options**.
- *OR* just use the 'Debug' options from the menubar and icon set.

Those wishing to use the CLI, seach online for '`gdb` tutorials' (certainly this is where I learned!).

- In Netbeans, debugging is launched with the icon **next to** the standard compile/run icon.

Go ahead and 'debug' your program. Does it complete? Consider that luck.

Now let's set a breakpoint:

1. Set a breakpoint at the `return` statement in `sum()` by left-clicking the line number.
2. Now launch debugging mode again.

Your code will pause at the breakpoint. Time to add watch variables. In the debugging window:

3. Click **Add Watch**.
4. Enter the variable name of interest. Start with **`c`**. What happens?
5. Repeat Steps 3 and 4 for each of **`x`**, **`y`**, **`z`**, **`*z`**. What do you see?

There should now be enough information to solve the problem. Good luck!

Additional Notes

It's rarely the case that the location of the crash is where the problem begins. One can set breakpoints earlier in the code and `step' their way through:

- **Next line** executes the next line of code *without entering functions.*
- **Enter Function** enters the function of interest, after which **Next Line** can be used within the function. Try this by setting a breakpoint at the call for `sum()`.