ECS 36A, May 15, 2023

Announcements

1. We haven’t graded the midterms yet; our target date is by Friday

Static Debugging

- Use printf's to print what is going on inside the program
  - Or anything else that prints
- You may need to print to stderr or file

sample1.c

- Goal is to add 2 to j 100 times
- But at end, j = 2
- Debug:
  - Put a printf right after line 16 (the j += 2); print both i and j
  - Note i = 100, j = 2
  - Put printf right before line 16 and have it print i and j
  - Oops . . . prints once
  - Clear j += 2 not in for loop
  - Look closely at line 16; note ending semicolon

sample1.c

- Goal is to add 2 to j 100 times
- But at end, j = 2
- Debug:
  - Put a printf right after line 16 (the j += 2); print both i and j
  - Note i = 100, j = 2
  - Put printf right before line 16 and have it print i and j
  - Oops . . . prints once
  - Clear j += 2 not in for loop
  - Look closely at line 16; note ending semicolon

sample2.c

- Goal is to double s until it is more than 100
- But it goes into an infinite loop, never ending
- Debug:
  - i is supposed to change to 0 in the if statement, so print s and i there
  - The printf statement recurs repeatedly, when j is more than 100
  - So we need to look at the value of i to see if it is reset to 1 somewhere
  - Print the value of i and s right after the while statement
  - It shows I is now 1, as we expect (otherwise we would not reenter the loop)
  - So check the while expression
  - In it, I is assigned 1; we need to make the = sign == for a test

sample3.c

- Goal is to count the number of digits, spaces, other characters
- But it prints 11 digit counts (last one is negative) and 0 blanks, and 828 others!
- Two bugs
  - An extra number printed with the digit counts
  - Everything but the ”other” has a count of 0 – and we had digits and spaces in the file

Extra Digit Bug

- The digits are printed in the for loop at the end
- It prints 11 but should print 10
- Print the values of i each time through the loop
- And i goes to 11, so examine the exit condition of the for loop
- It should stop when i is 10, but instead prints a value then
- Change the “<=“ to “<“ (or the ”10” into a “9”) to solve this

All Characters are “Other”

- The characters are classified in the while loop, so it’s clear the char c is not set properly
- Print the value of c right after the while statement
- Output (c is ‘’) looks like c isn’t read; this is a non-printing character
- Print the value of c as both a character *and* a hexadecimal number
- It comes out as 0x01 so c is being set to 1
- It’s assigned in the while expression, so we look there
- Note != has higher precedence than =, so while chars are read, getchar() returns 1 as the condition is true, and so c is assigned 1.
- To fix this, use parenthesis: “(c = getchar()) != EOF”

gdb

- A dynamic debugger
- To run it, compile your program with the `–g` option
  - This adds in debugging information *gdb* uses
  - You can use *gdb* without it but it simplifies the use greatly
- Then load it into *gdb* by:

```
gdb executable
```

- Note you use the executable file and *not* the source code file
  - You can also load the executable once *gdb* starts

Inside the *gdb* Shell

- Once started, you get a prompt "(gdb)"
- If you forgot to name the executable in the command line:

```
(gdb) file executable
```

- One other handy feature

```
(gdb) help
```

- You will get a list of commands you can ask for help on
- Then type

```
(gdb) help command
```

Executing the program

- Type:

(gdb) run arg_1 . . . arg_n

- This runs the program with command line arguments arg_1 through arg_n
  - If there are no command line arguments, just type ``run''
- If there are no problems, the program runs to completion
- If the program stop with a message like this, there's a problem

```
Program received signal SIGSEGV, Segmentation fault.
0x00005555555551b5 in nfact (n=<error reading variable: Cannot access
memory at address 0x7fffff7fefec>) at nfact2.c:12
```

Stopping the Program Before It Ends

- A *breakpoint* causes the execution to stop at that point
- Here's an example:

```
(gdb) break 15
Breakpoint 1 at 0x5555555551b8: file nfact2.c, line 15.
```

- This causes execution to stop when it reaches line 15
  - If you have multiple source files, name the file before the number:

```
(gdb) break nfact2.c:15
```

- It shows some useful information

```
Breakpoint 1, nfact (n=15) at nfact2.c:15
15 x = nfact(n+1);
```