CSE 351: The Hardware/Software Interface

Section 3

Control flow, assembly, lab 2

Advanced control flow

- Let’s look at some less-common control flow operators and review how to use them
- For each control flow operator, we will examine the assembly code and see how it relates
- See the schedule page for the accompanying section material (http://www.cs.washington.edu/education/courses/cse351/13wi/schedule.html)

do-while (see dowhile.c)

- do-while loops are useful when the exit condition is only relevant after executing the body of the loop once

```
int value;
do {
  value = computeSomething(value);
} while (value != 10);
```

switch-case (see switchcase.c)

- switch-case blocks are useful when there are a fixed number of values that a variable can have, each of which should be handled separately
- How does the efficiency of a switch-case compare to if-else if-else?

```
int computeSomething(int value) {
  switch (value) {
    case 0:
    case 1:
      value = value + 2;
      break;
    case 2:
      value = value + 3;
      break;
    default:
      ++value;
  }
  return value;
}
```

switch-case (see switchcase.c)

- In this example, if value is either 0 or 1, the statement “value = value + 2;” will be executed and then “break;” will exit the block
- In the absence of “break;”, code execution will “fall through”

```
int computeSomething(int value) {
  switch (value) {
    case 0:
    case 1:
      value = value + 2;
      // break; <- after commenting this out, execution will proceed
      //           through the “case 2” logic as well.
    case 2:
      value = value + 3;
      break;
    default:
      ++value;
  }
  return value;
}
```

ternaries (see ternaries.c)

- Ternaries are extremely handy for expressing concise if-else relations
- Use: `condition ? true-value : false-value;`

```
int getValue(int* ptr) {
  // return 0 if ptr is NULL, otherwise
  // the value it points to.
  return ptr == NULL ? 0 : *ptr;
}
```

goto (see goto.c)

- gotos are useful for error handling and some other special cases, but should otherwise be avoided if possible (code becomes far less readable)

```
int computeSomething(int value) {
start:
  ++value;
  if (value % 5 == 0)
    goto end;
  else
    goto start;
end:
  return value;
}
```

Lab 2

- Use GDB, objdump, and other tools to figure out code words to defuse the bomb
- The files involved:
  - bomb: An executable bomb file. Takes code phrases on separate lines as input
  - bomb.c: Defines the entry point of the program. Calls functions whose source code is not available to you
  - defuser.txt: Contains pass phrases for each stage, separated by newlines. Add each pass phrase here as you discover it

GDB with lab 2

- GDB allows you to see the assembly code for functions, view the contents of registers, and set breakpoints to look at values at particular locations
- Sample workflow:

```
$ gdb --args ./bomb defuser.txt
(gdb) start # start the program (enter main method)
(gdb) b [function-or-address] # set a breakpoint
(gdb) c # continue execution of the code
(GDB will hit the breakpoint)
(gdb) info registers # look at register values
(gdb) disassemble # print assembly code
(gdb) stepi # step one instruction
(gdb) nexti # step one instruction, skipping calls
(gdb) c # start executing again
```

objdump and strings with lab 2

- `objdump -t` lets you see the symbols contained in the bomb file, e.g. `objdump -t bomb`
  - Which symbols correspond to functions? Which functions are specific to the bomb code as opposed to the GNU C library?
- `strings -t x bomb` will print out the readable strings contained in the bomb file
  - Does the output contain anything useful?

Lab 2 notes

- Each student in the class has a different bomb; no two have the same answers
- Make sure to put the pass phrases you discover in the defuser.txt file so that you don’t have to type them in each time
- GDB has built-in help for all of its functions
  - (gdb) help info
  - (gdb) help disassemble
- Can also search online for help with GDB

Lab 2 notes

- The bomb makes use of sscanf, which parses a string into values
- As an example:

```
int a, b;
sscanf("123, 456", "%d, %d", &a, &b);
```

- The first string is parsed according to the format string of the second argument
- Upon success, the values of a and b will be set to 123 and 456, respectively
- Refer to `man 3 sscanf` for more information