Week 3 - Friday

COMP 2400

Last time

- What did we talk about last time?
- Control flow
- Selection
  - `if` statements
  - `switch` statements
- Loops
  - `while`
  - `for`
  - `do-while`
- Common loop errors

Questions?

Project 2

Quotes

Unix was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things.

Doug Gwyn

Bad Things

break

- The **break** command is a necessary part of the functioning of a **switch** statement
- But, it can also be used to jump out of a loop
- Whenever possible (i.e. always), it should not be used to jump out of a loop
  - Everyone once in a while, it can make things a little clearer, but usually not
  - Loops should have one entry point and one exit

```
for (int value = 3; value < 1000; value += 2)
{
        …
        if (!isPrime(value))
                break;
}
```

```
for (int value = 3; value < 1000 && isPrime(value); value += 2)
{
        …
}
```

`continue`

- The **`continue`** command is similar to the **`break`** command
- It will cause execution to jump to the bottom of the loop
- If it is a **`for`** loop, it will execute the increment
- For all loops, it will return to the top if the condition is true
- It makes things easier for the programmer up front, but the code becomes harder to follow
- The effect can be simulated with careful use of **`if`** statements

goto (a four letter word)

- A **goto** command jumps immediately to the named label
- Unlike break and continue, it is not a legal command in Java
- Except in cases of extreme (**EXTREME**) performance tuning, it should never be used
  - Spaghetti code results

```
for (int value = 3; value < 1000; value += 2)
{
    if (!isPrime(value))
        goto stop;
}
printf("Loop exited normally.\n");
stop:
printf("Program is done.\n");
```

Loop practice

- Read in a series of numbers and output the smallest

More loop practice

- Write a loop that counts the number of digits in a number
- Hint: Keep dividing the number by 10 until you get 0

Even more loop practice

- A regular number is one divisible by only 2, 3, and 5
- Print out the first 50 regular numbers:
  - 1 2 3 4 5 6 8 9 10 …

Systems Programming

System calls

- A **system call** is a way to ask the kernel to do something
- Since a lot of interesting things can only be done by the kernel, system calls must be provided to programmers via an API
- When making a system call, the processor changes from user mode to kernel mode
- There's a fixed number of system calls defined for a given system

`glibc`

- The most common implementation of the Standard C Library is the GNU C Library or **`glibc`**
- Some of the functions in the **`glibc`** perform systems calls and some do not
- There are slight differences between the versions of the **`glibc`**
  - Microsoft also has an implementation of the Standard C Library that doesn't always behave the same

Screen output

- It turns out that there are two kinds of output to the terminal
  - **`stdout`** (where everything has gone so far)
  - **`stderr`** (which also goes to the screen, but can be redirected to a different place)
- The easiest way to use **`stderr`** is with **`fprintf()`**, which can specify where to print stuff

```
fprintf(stderr, "Going to stderr\n!");
printf("Going to stdout\n!");
```

Redirecting streams

- When you redirect **stdout**, **stderr** still goes to the screen

```
./program > out.file
Going to stderr.
```

- If you want to redirect **stderr** to a file, you can do that as well with **2>**

```
./program > out.file 2> error.log
```

Handling system errors

- There are no exceptions in C
- Instead, when a system call fails, it usually returns **`-1`**
- To find out why the system call failed
  - First, make sure you **`#include <errno.h>`**
  - Then check the value of the integer **`errno`** in your program after the system call fails
  - Use the man pages to determine what a given value of **`errno`** means
- The **`perror()`** function is often used to print errors instead of **`printf()`**
  - It sends the output to **`stderr`** instead of **`stdout`** and then prints a message based on **`errno`**

Error handling example

```
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
int main(){
      int fd = open("eggplant.txt", O_WRONLY | O_CREAT | O_EXCL);
      if (fd == -1) {
             perror("Failure to create file");
             if(errno == EACCES)
                    fprintf(stderr, "Insufficient privileges\n");
             else if(errno == EEXIST)
                    fprintf(stderr, "File already exists\n");
             else
                    fprintf(stderr, "Unknown error\n");
             exit(EXIT_FAILURE);
      }
      return 0;
}
```

System types

- C has a feature called **`typedef`** which allows a user to give a new name to a type
- System types are often created so that code is portable across different systems
- A common example is **`size_t`**, which is the type that specifies length
  - It's usually the same as **`unsigned int`**
- There are named types for process IDs (**`pid_t`**), group IDs (**`gid_t`**), user IDs (**`uid_t`**), time (**`time_t`**), and many others

Upcoming

Next time…

- Functions

Reminders

- Read K&R chapter 4
- Keep working on Project 2