CS 521: Systems Programming

C Data Types, Command Line Arguments

Lecture 5

Today's Schedule

- Phases of Compilation
- Data Types
- C Input/Output: `echo`

Today's Schedule

- **Phases of Compilation**
- Data Types
- C Input/Output: `echo`

Compiling Your Programs

- You might have not cared much about compiling code previously
  - **Compile**: turn code into an executable
- …but with C, it's a bigger deal
- The C compiler goes through a few phases to get from **code** to a finished, ready-to-run **binary executable**

Phases of C Compilation

1. **Preprocessing**: perform text substitution, include files, and define macros. The first pass of compilation.
   - Directives begin with a #
2. **Translation**: preprocessed code is converted to machine language (also known as ***object code***)
3. **Linking**: adding external routines (for example, `printf` from stdio.h).
   - Sometimes you'll compile separate modules to **object files** (.o) and link them to form a single binary

Today's Schedule

- Phases of Compilation
- **Data Types**
- C Input/Output: `echo`

C Data Types

- When defining arguments and variables, the following data types are possible in C:
  - `char`
  - `int`
  - `float`
  - `double`
- Wait… that’s it?! Yeah! Well, there are a few ***modifiers***:
  - `short`, `long`, `signed`, and `unsigned`

Sizing

- `short` and `long` modify the data type's size
- The C standard specifies the *minimum* size for each type. You can determine the sizes (in bytes) with `sizeof`:
  - `sizeof(char) = 1`
  - `sizeof(short int) = 2`
  - `sizeof(int) = 4`
  - `sizeof(long int) = 8`
- ...but these can be platform-specific. Don't make assumptions!
  - One thing can be certain: `char` is **guaranteed** to be 1 byte

Demo: Data Type Sizes

(you can do this one on your VM, or local machine if you have a C compiler!)

Signed Data Types

- Integer types can be **signed** or **unsigned**
  - Signed integers use one bit as a *sign bit* to determine whether the number is negative or positive
- Java doesn't have unsigned `ints`. What might they be useful for?
  - Enforce a particular variable to always be positive
  - Use that extra bit to store larger positive numbers
- Related: integer overflow is undefined behavior (UB)

Today's Schedule

- Phases of Compilation
- Data Types
- **C Input/Output: `echo`**

Creating an Echo Chamber

- To demonstrate C input and output (I/O), we'll write a program that takes input strings… and then outputs them!
- There's already a utility that does this: `echo`
    - Let's use a project-based approach to make our own
- Hear that?
    - `echo`
        - `echo`
            - `echo`
                - `echo`

Echo

- What does the `echo` command do?

```
[mmalensek@mmalensek-vm ~]$ echo

```

Wow!!!

```
[mmalensek@mmalensek-vm ~]$ echo Hello World!
Hello World!
```

Going to the Documentation

- You probably already have a good grasp of what echo does, but let's go to the ***real*** authority: the documentation!
- To access the **manual pages**, use the `man` command
- `man echo`

Gathering Requirements

- What do we need to be able to do to build our own copy of `echo`?
- The **GNU** version of `echo` supports a ton of features… Maybe we can copy the **BSD** version instead
  - (command line tools have a standard set of features, but there are several different implementations!)
- Take a few minutes to come up with requirements…

Requirements

Here’s what I came up with.

- A way of accessing the *command line arguments* passed to the program (e.g., `./prog arg1 arg2 arg3`)
- A loop so we can iterate through each one
- We already know how to print… sort of. More detail there would be good
- We need to handle the `-n` *command line flag*

Command Line Arguments

- In Java, the `main` method has one argument: an array of strings that contain the command line args
- So far we've seen one way of declaring `main` in C:

  `int main(void)`
- There is **another way** to do it!

  `int main(int argc, char *argv[])`
  - `argc`: argument count
  - `argv`: argument values (as an array of `char *`… what's that?)

The First Argument

- The **first** argument will always be the *program name*
- i.e., if you run `./some_prog` then `argv[0] = "./some prog"`
- This also means that `argc` will always be at least **1**

Next Requirement: A Loop

- We can use a `for` loop with the `argc` count to loop through all the arguments
- We haven't fully discussed arrays yet, but let's just pretend we know what we're doing!
- If I access `argv[i]` I will get the i^{th} value of the array of… `char *`?

What the $%*@ is char star?

- In C, the `*` indicates a *pointer*. So a `char *` type is a **pointer to a character**.
- C does not have a string type… instead, we use *arrays of characters*
- So `char *argv[]` is an array of pointers to characters
  - geez
- Understanding that seems like it might take work, so let's save that for another day…

Printing

- We can google how to use `printf`, and we'll get some great answers
- But we can also look at the documentation:
- `man 3 printf`
  - `man 3` means use the 3rd section of the manual – the C documentation.
  - `man printf` will actually give you information about something else – the `printf` command line utility

Printing a String

- We can use `printf("%s", some_string);` to print a string
- If we use `puts(some_string)` it will include a **newline** character (`\n`) at the end, and we don’t want that

Handling Flags

- Most command line utilities support **flags** to make them behave in different ways
- When `echo` receives a `-n` flag, it doesn't print a trailing newline
- How can we handle this? With a conditional!

  -
    ```
    if (argv[i][0] == '-') {
    /* First letter is a - character! */
    /* What do we check for next? */
    }
    ```

WAIT!

- I thought `argv` was an array of pointers to **A** character, right?
  - How are we indexing into it twice like a 2D array?
- Well…
- This is because in C, strings are arrays of characters.
  - When you create a string, it is represented as a pointer to the first character in that string
- When we do `argv[i][0]` we are accessing the first character in the string
- Weird, but don't worry yet. We will talk about Strings a LOT more

Putting it Together

- We have enough information to start building an `echo` utility
- This is the first part of Lab 2.
- Next up: more strings and I/O!