CIS 190: C/C++ Programming

Lecture 1

Introduction and Getting Started

This course will…

- teach you the basics of C and C++
- give you more programming experience
- be appropriate for majors and non-majors
- *not* make you an expert in C or C++
- *not*, by itself, make you ready to take on a C/C++ programming job, or design and write a professional C/C++ application
- enable you to learn more independently

Why learn C/C++?

- ubiquitous in software engineering today and into the future
  - see the Tiobe Index
- helps you understand computers better
- “close to the metal”
- why start with C instead of C++?
  - “under the hood”

Word of warning

“C is a language that values speed and programmer control over guaranteed safety. You are expected to develop habits that will keep you safe rather than doing random things and hoping to be caught.”

– Kate Gregory

Administrivia

- grades based on homework (70%) and final project (30%) curved at end of semester
- this is “recitation” but it’s actually the class
- Academic Honesty:
  - work must be **yours**
  - do **not** share code

Getting started

- editor
  - xemacs, emacs, pico, vim
  - setup guide for Windows machines on website
- compiler
  - run in terminal; gcc
- using libraries: #include <libname.h>
  - stdio.h, string.h, stdlib.h
- C and C++ require main()

hello_world.c

```
#include <stdio.h>

int main()
{
    printf("Hello world!\n");

    return 0;
}
```

Compiling with C: gcc

- to compile a file to an executable called hello:

```
gcc hello_world.c –o hello
```

- to run the executable output:

```
./hello
```

- to compile a file with warnings:

```
gcc hello_world.c –Wall
    -o hello
```

Anatomy of hello_world.c

```
/* includes functions from the
   standard library, like printf */
#include <stdio.h>

/* main – entry point to our program */
int main()
{
   /* prints “Hello World!” and a newline to screen*/
   printf(“Hello world!\n”);

   /* returns the result code for the
      program – 0 means there was no error */
   return 0;
}
```

printf

- **`printf(<format string>`**`,` **`<arguments>)`**`;`
  - prints the given format string to the console
- <format string> is text you want to print and specifiers (like **`%s`**) for additional arguments
- the <arguments> are handled in order

- unlike **`System.out.println`**, need to insert line breaks manually: **`\n`**

printf format string specifiers

- there are lots – the basic ones:
  - %d: int (e.g., 7)
  - %c: char (e.g., ‘a’)
  - %s: string (e.g., “hello”)
  - %f: float (e.g., 6.5)

```
printf(“It was %f %s at %d%c.\n”, 72.5, “degrees”,
       221, ‘B’);
> It was 72.5 degrees at 221B.
```

Precision with printf

- printf can also format the text it prints out:

```
printf(“_%4d_ or %6s or _%-6s_”, 18, “dog”, “a”);
> _  18_ or _   dog_ or _a     _
```

  – place the precision number after the %

  – for floats/doubles, you can specify both before and after the decimal: `%4.2f`

- spaces take up extra <precision> characters
- use a negative sign to left-align

scanf

- reads information from the console (user)
- need to know details about input (formatting)
- ignores whitespace characters
- information read in is stored in a variable, referenced by a pointer to that variable

```
int x;
scanf("%d", &x);
```

- same format specifiers as printf

for loops and local variables

- for loops work as expected, but local variables must be declared at the start of the function

```
int main() {
  int i;
  for (i = 0; i < 14; i++) {
    printf("%d ", i);
  }
}
> 0 1 2 3 4 5 6 7 8 9 10 11 12 13
```

Variable types in C

- unlike Java, there is no boolean or String
  - in C, 0 = false, and everything else = true
  - variables are also not initialized for you!
- int, char, float, double… long, short, etc.
  - signed and unsigned
- also, there are no vectors, only arrays
  - there is no ‘new’ keyword either

Arrays in C

- like in Java:
  - Declaration: <type> <name> [size];

    ```
    int xArray [10]; /* creates array of 10 ints */
    ```

  - Indexing:
    - starts at 0: `xArray[4]; /* 5th element*/`
- unlike Java:
  - no bounds checking/built-in knowledge of size
  - can't return an array from a function!

Declaring arrays in C

- arrays are static
- once they're created, their size is fixed

- size needs to be pre-determined at compile time
  - for example, this means you cannot make it a value that is read in from the user
  - **unless** you allocate the memory manually

EXAMPLE: Sieve of Eratosthenes

- algorithm to quickly identify prime numbers
- first, assume all numbers >= 2 are prime:

```
x x 2 3 4 5 6 7 8 9 10 11 12 13
```

- then, take the first prime number and mark all of its multiples as composite:

```
0 1 2 3 x 5 x 7 x 9 x 11 x 13
```

- and so on:

```
0 1 2 3 4 5 x 7 8 x 10 11 x 13
```

Functions in C

- similar to in Java, but not exactly:
- <return value> <name> (<parameters>)
  - no “public static” necessary

- three “parts” to using a function in C:
  - function prototype
  - function definition
  - function call

Function Parts

- Prototype:
  - function must be declared before it can be used

```
int SquareNumber (int n);
```

- Definition:
  - function must be defined

```
int SquareNumber (int n) { return (n * n); }
```

- Call:

```
int answer, input = 3;
answer = SquareNumber(input);
```

Functions in hello_world.c

```
#include <stdio.h>
void PrintHelloWorld();    /* function prototype */

int main()
{
   PrintHelloWorld();      /* function call */
   return 0;
}

void PrintHelloWorld()     /* function definition */
{
   printf(“Hello world!\n”);
}
```

C-style strings

- no native “String” type in C
- represented as an array of characters
  - terminated by the NULL character, ‘`\0`’
- a C-string joke:
  - Two strings walk into a bar.
  - The bartender says, "What'll it be?".
  - The first string says, "I'll have a gin and tonic#MV*()>SDk+!^��&�@P&]JEA&#65535".
  - The second string says, "You'll have to excuse my friend, he's not null-terminated."

C-style strings

- three ways to declare a string (statically)

```
char *str1     = “dog”;
char  str2 [] = “cat”;
char  str3 [5];
```

- first two ways require initial value; length is set at that initial string’s length
- third way creates string of length 5

C strings are arrays of characters

char *str1 = “dog”;					
element	0	1	2	3	
char	‘d’	‘o’	‘g’	‘\0’	
char str2 [] = “cat”;					
element	0	1	2	3	
char	‘c’	‘a’	‘t’	‘\0’	
char str3 [5];					
element	0	1	2	3	4
char	‘.’	‘N’	‘=’	‘¿’	‘8’

- str3 was only declared, not initialized, so it’s filled with garbage and has no null terminator

C strings are arrays!

- just like you can't compare two whole arrays, you can't just compare strings
  - **`str1 == str2`** will not do what you think
- library of string functions – #include <string.h>
  - strcmp will compare two strings:

    ```
    int same = strcmp(str1, str2);
    ```
  - strcpy will copy the second string into the first

    ```
    strcpy(str1, "success!");
    ```

Homework 1

- “Hello, World!”
  - www.cis.upenn.edu/~cis190
- make sure that you have a working setup
  - SEAS account
  - editor
  - **eniac.seas.upenn.edu** compiler
- check that you can submit – no mercy!

- hello_world.c, answers.txt
  - take credit for your code – comment at top!