CIS 190: C/C++ Programming

Lecture 1

Introduction and Getting Started

Outline

- Syllabus
- Quiz
- Intro to C Programming
  - Basics
  - Sieve of Eratosthenes example
  - More Basics
- Homework

This course will…

- teach you the basics of C and C++
  - C will give you more “under the hood” experience
  - C++ is more powerful, but gives less control
- give you more programming experience
- enable you to learn more independently

This course will not…

- make you an expert in C or C++
- by itself, make you ready to take on a C/C++ programming job, or design and write a professional C/C++ application
- cover the basics (boolean logic, loops, recursion, = VS ==, etc.)

Lecture/Recitation

- the “lecture” portion is shared by all CIS 19X
- this is the “recitation” – it’s the actual class
  – homeworks, projects, and your grade
- there is no required textbook
  – stackoverflow.com can be very helpful

Grades

- grades are based on:
  - homework (70%)
  - final project (30%)
- no exams
- grades curved at end of semester
  - available throughout semester on Canvas

Homeworks

- due roughly weekly

- coding environment is left to you, but homeworks ***must run*** correctly on the eniac.seas.upenn.edu machines

- challenge problems are optional, but are more work than the points given might indicate

Late Days

- you have 3 late days for use on homeworks
  - can use up to 2 days for a single homework
  - can't use late days for project deliverables
- late days take effect ***immediately*** after the due date
- can track how many are left on Canvas

Final Project

- timeline of about a month
- design and implement your own choice of project in C++
- must work in pairs

- past projects include implementations of Solitaire and Clue, text-based adventures, and helper apps like to-do lists

Policies – Academic Honesty

- all work must be yours
- do not share code with classmates
- do not plagiarize code from the Internet

Policies – Coding Style

- coding style/standards are part of industry
- good programmers know how to adhere to a coding style
- reading the coding style on the webpage is part of your first homework

Policies – Coding Style

- don’t use “magic numbers”
- don’t use global variables
- don’t use while(1) loops

- do comment your code and your files
- do use meaningful names
- do be consistent with your brace style

Policies – Attendance

- attendance is not mandatory

- there are significant advantages though:
  - ask questions in real time
  - input on what topics we cover
  - hints and tips for homeworks

Policies – Contact

- use the Piazza page for any questions about the course or assignments
  - use "private" questions if you need to post code
  - don't email the TAs/Instructor
- check the website for updates on lectures, homework, etc.
- Office Hours will be announced soon

Outline

- Syllabus
- Quiz
- Intro to C Programming
  - Basics
  - Sieve of Eratosthenes example
  - More Basics
- Homework

Quiz

Outline

- Syllabus
- Quiz
- Intro to C Programming
  - Basics
  - Sieve of Eratosthenes example
  - More Basics
- Homework

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

Basics

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

hello_world.c

```
#include <stdio.h>

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

    return 0;
}
```

LIVECODING

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
```

LIVECODING

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;
}
```

LIVECODING

Variable types in C

- cannot assume variables will be initialized

- variables available in C:
  - int, char, float , double, etc.
  - long/short, signed/unsigned, etc.
- not available in C:
  - boolean, String

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

printf format string specifiers

- some of 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 example

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

```
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("_%5d_ or %4s or _%-3s_",
         18, "dog", "a");
> _   18_ or _ dog_ or _a  _
```

- spaces used to pad out any extra characters
- use a negative sign to left-align

Precision with printf

- for floats/doubles, you can specify precision for both before and after the decimal: `%3.6f`
  - place the precision number right after the `%`

```
printf("your grade is: %2.3f",
         92.3333333);
> your grade is: 92.333
```

scanf

- reads information from the console (user)
- need to know details about input (formatting)

- ignores whitespace characters
  - spaces, tabs, newlines

- uses same specifiers as printf (**`%d`**, **`%s`**, etc.)

Storing information from scanf

- information read in is stored in a variable

```
/* int, float, and char take pointers */
scanf(“%d”, &int_var);
scanf(“%f”, &float_var);
/* char does not do a good job of
   skipping whitespace */
scanf(“ %c”, &char_var);
/* string does not take a pointer */
scanf(“%s”, string_var);
```

scanf example

```
int x;
printf(“Enter value for x: ”);
scanf(“%d”, &x);
printf(“x is %d\n”, x);

> ./a.out
> Enter value for x:
```

scanf example

```
int x;
printf("Enter value for x: ");
scanf("%d", &x);
printf("x is %d\n", x);

> ./a.out
> Enter value for x: 18
```

scanf example

```
int x;
printf(“Enter value for x: ”);
scanf(“%d”, &x);
printf(“x is %d\n”, x);

> ./a.out
> Enter value for x: 18
> x is 18
```

Arrays in C

- Declaration:

```
<type> <name> [size];
float xArray [10];
```

- Indexing starts at 0:

```
xArray[9]; /* end of the array */
```

Limitations of Arrays

- no bounds checking
- no built-in knowledge of array size
- can’t return an array from a function

- arrays are static
  - once created, their size is fixed

Declaring arrays in C

- 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 (which we’ll cover in later lectures)

for loops and local variables

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

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

Outline

- Syllabus
- Quiz
- Intro to C Programming
  - Basics
  - Sieve of Eratosthenes example
  - More Basics
- Homework

Sieve of Eratosthenes

- algorithm to quickly identify prime numbers

- before starting, choose the highest number you’d like to check for being prime
- assume all numbers >= 2 are prime numbers
- for each prime number you encounter, mark all of its multiples as composite (not prime)

EXAMPLE: Sieve of Eratosthenes

- 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
```

LIVECODING

Outline

- Syllabus
- Quiz
- Intro to C Programming
  - Basics
  - Sieve of Eratosthenes example
  - More Basics
- Homework

Functions in C

- `<return value> <name> (<parameters>)`
  - no “public” or “static” necessary
- there are 3 “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 = SquareNumber(3);
```

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”);
}
```

LIVECODING

C-style strings

- there is no native “String” type in C

- instead, strings in C are represented as an array of characters
  - terminated by the NULL character, ‘`\0`’
  - (backslash zero)

Declaring C-style strings

- three ways to (statically) declare a string

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

- first two ways require initial value; length is set at that initial string’s length (i.e., 4)
- third way creates string of length 5

C strings are arrays of characters

`char *str1 = “dog”;`					
element	0	1	2	3	
char					
`char str2 [] = “cat”;`					
element	0	1	2	3	
char					
`char str3 [5];`					
element	0	1	2	3	4
char					

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					

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					

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

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

Terrible C-style 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 are arrays!

- you can't compare two arrays using **`==`**
- so you can't compare strings that way either

`str1 == str2` will <u>not</u> do what you think

- also can't assign one array to another using =
- so you can't assign one string to another

`str1 = str2` will <u>not</u> do what you think

C-style string library functions

- to use you must `#include <string.h>`

- `strcmp` will compare two strings:

```
int notSame = strcmp(str1, str2);
```

  – returns a “0” if the string are identical, not a 1!

- `strcpy` will copy the 2nd string into the 1st

```
strcpy(str1, “success!”);
```

Outline

- Syllabus
- Quiz
- Intro to C Programming
  - Basics
  - Sieve of Eratosthenes example
  - More Basics
- Homework

Homework 1

- five things to do:
  - complete hello_world.c
  - complete answers.txt
  - turn the above two files in using **`turnin`**
  - read the style guide online
  - fill out availability for OH on when2meet
    - use your upenn username