CIS 190: C/C++ Programming

Lecture 6

Introduction to C++

Outline

- Changes for C++
  - Files & Compiling
  - Variables
  - Functions
- Input/Output in C++
  - cin/cout/cerr
  - Print Functions
  - Reading/Writing to Files
- hello_world.cpp

Files in C++

- **`hello_world.c`**

Files in C++

- **`hello_world.c`**
  - becomes
- **`hello_world.cpp`**

Files in C++

- **`hello_world.c`**
  - becomes
- **`hello_world.cpp`**

- **`hello_world.h`**

Files in C++

- **`hello_world.c`**
  - becomes
- **`hello_world.cpp`**

- **`hello_world.h`**
  - stays
- **`hello_world.h`**

Compiling in C++

- instead of **`gcc`** use **`g++`**

- you can still use the same flags:
  - **`-Wall`** for all warnings
  - **`-c`** for denoting separate compilation
  - **`-o`** for naming an executable
  - **`-g`** for allowing use of a debugger
    - and any other flags you used with gcc

Outline

- Changes for C++
  - Files & Compiling
  - Variables
  - Functions
- Input/Output in C++
  - cin/cout/cerr
  - Print Functions
  - Reading/Writing to Files
- hello_world.cpp

Variables in C++

- comments can be
  ```
  /* contained with asterisks */
  ```
  or
  ```
  // all text after is a comment
  ```

- **`#define`** will still work
  - but we can also use **`const`** instead

#define vs const

- **`#define`** replaces with value at compile time

```
#define PI 3.14159265358979
int main()
{
  printf("Pi is %f\n",
         PI);
}
```

#define vs const

- **`#define`** replaces with value at compile time

```
#define PI 3.14159265358979
int main()
{
  printf("Pi is %f\n",
         3.14159265358979);
}
```

#define vs const

- **`const`** defines variable as unable to be changed

```
const double PI = 3.14159265358979;
```

- regardless of the choice, they are used the same way in code

```
area = PI * (radius * radius);
```

Details about const

```
const double PI = 3.14159265358979;
```

- explicitly specify actual type
- a variable – so can be examined by debugger

- const should not be global
  - very very rarely
  - normally used inside classes

Interacting with Variables in C

- in C, most of the variables we use are "primitive" variables (int, char, double, etc.)

- when we interact with primitive variables using provided libraries, we call functions and pass those variables in as arguments

```
fopen(ifp, "input.txt", "r");
free(intArray);
strlen(string1);
```

Interacting with Variables in C++

- in C++, many of the variables we use are instances of a class (like string, ifstream, etc.)

- when we want to interact with these variables, we use method calls on those variables

```
inStream.open("input.txt");
string2.size();
```

Using Variables in C++

- declaration is more lenient
  - variables can be declared anywhere in the code
  - may still want them at the top, for clarity

- C++ introduces new variables
  - string
  - bool

string

- requires header file: `#include <string>`

Some advantages over C-style strings:

- length of string is not fixed
  - or required to be dynamically allocated
- can use “normal” operations
- lots of helper functions

Creating and Initializing a string

- create and initialize as empty

```
string name0;
```

Creating and Initializing a string

- create and initialize as empty

```
string name0;
```

- create and initialize with character sequence

```
string name1 ("Alice");
string name2 = "Bob";
```

Creating and Initializing a string

- create and initialize as empty

```
string name0;
```

- create and initialize with character sequence

```
string name1 ("Alice");
string name2 = "Bob";
```

- create and initialize as copy of another string

```
string name3 (name1);
string name4 = name2;
```

“Normal” string Operations

- determine length of string

```
name1.size();
```

- determine if string is empty

```
name2.empty();
```

- can use the equality operator

```
if (name1 == name2)
```

More string Comparisons

- can also use the other comparison operators:

```
if (name1 != name2)
```

- alphabetically (but uses ASCII values)

```
if (name3 < name 4)
if (name3 > name 4)
```

- and can concatenate using the '**+**' operator

```
name0 = name1 + " " + name2;
```

Looking at Sub-Strings

- can access one character like C-style strings

```
name1[0] = 'a';
```

- can access a sub-string

```
name1.substr(2,3);
```

  - "ice"

```
name2.substr(0,2);
```

  - "Bo"

bool

- two ways to create and initialize

```
bool boolVar1 = true;
bool boolVar2 (false);
```

- can compare (and set) to true or false

Outline

- Changes for C++
  - Files & Compiling
  - Variables
  - Functions
- Input/Output in C++
  - cin/cout/cerr
  - Print Functions
  - Reading/Writing to Files
- hello_world.cpp

Functions in C++

- some similarity to functions in C
  - variables are only in scope within the function
  - require a prototype and a definition
  - arguments can still be passed by reference or passed by value

- one small difference: no need to pass array length (can just use empty brackets)

```
void PrintArray (int arr []);
```

Using `const` in C++ functions

- when used on pass-by-value

```
int SquareNum (const int x) {
  return (x * x);  /* fine */
}

int SquareNum (int x) {
  return (x * x);  /* fine */
}
```

Using `const` in C++ functions

- when used on pass-by-value

- no real difference; kind of pointless
  - changes to pass-by-value variables don't last beyond the scope of the function

- **conventionally**: not "wrong," but not done

Using `const` in C++ functions

- when used on pass-by-reference

```
void SquareNum (const int *x) {
   (*x) = (*x) * (*x);  /* error */
}

void SquareNum (int *x) {
   (*x) = (*x) * (*x);  /* fine */
}
```

Using `const` in C++ functions

- when you compile the “const” version:

```
void SquareNum (const int *x) {
   (*x) = (*x) * (*x);  /* error */
}
```

```
error: assignment of read-only
       location '*x'
```

Using `const` in C++ functions

- when used on pass-by-reference
- prevents changes to variables, even when they are passed in by reference

- **conventionally**:
  - use for user-defined types (structs, etc.)
  - don't use for simple built-in types (int, float, char)
    - except maybe arrays

Outline

- Changes for C++
  - Files & Compiling
  - Variables
  - Functions
- Input/Output in C++
  - cin/cout/cerr
  - Print Functions
  - Reading/Writing to Files
- hello_world.cpp

Working with Input/Output in C++

- at top of each file that uses input/output

```
using namespace std;
```

- to use streams to interact with user/console, must have `#include <iostream>`

- to use streams to interact with files, must have `#include <fstream>`

Input/Output in C++

```
#include <stdio.h>


printf("test: %d\n", x);


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

Input/Output in C++

```
#include <stdio.h>
#include <iostream>

printf("test: %d\n", x);


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

Input/Output in C++

```
#include <stdio.h>
#include <iostream>
using namespace std;
printf("test: %d\n", x);


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

Input/Output in C++

```
#include <stdio.h>
#include <iostream>
using namespace std;
printf("test: %d\n", x);
cout << "test: " << x << endl;

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

Input/Output in C++

```
#include <stdio.h>
#include <iostream>
using namespace std;
printf("test: %d\n", x);
cout << "test: " << x << endl;

scanf("%d", &x);
cin >> x;
```

The << Operator

- insertion operator; used along with **cout**

- separate each "type" of thing we print out

```
int x = 3;
cout << "X is: " << x
     << "; squared "
     << SquareNum(x) << endl;
```

The << Operator

- insertion operator; used along with **cout**

- separate each "type" of thing we print out

```
int x = 3;
cout << "X is: " << x
     << "; squared"
     << SquareNum(x) << endl;
```

The `>>` Operator

- extraction operator; used with **`cin`**
  - returns a boolean for (un)successful read

- like scanf and fscanf, skips leading whitespace, and stops reading at next whitespace
- don't need to use ampersand on variables

```
cin >> firstName >> lastName >> age;
```

using namespace std

- at top of each file you must have

```
using namespace std;
```

- otherwise you must use instead of

```
std::cin        cin
std::cout       cout
std::endl       endl
```

cerr

- in addition to **`cin`** and **`cout`**,
  we also have a stream called **`cerr`**

- use it instead of **`stderr`**:

```
fprintf(stderr, "error!\n");
```

cerr

- in addition to **`cin`** and **`cout`**,
  we also have a stream called **`cerr`**

- use it instead of **`stderr`**:

```
fprintf(stderr, "error!\n");
cerr << "error!" << endl;
```

Outline

- Changes for C++
  - Files & Compiling
  - Variables
  - Functions
- Input/Output in C++
  - cin/cout/cerr
  - Print Functions
  - Reading/Writing to Files
- hello_world.cpp

Quick Note on “Print” Functions

two basic ways to handle printing:

- function returns a string

- function performs its own printing

Quick Note on “Print” Functions

two basic ways to handle printing:

- function returns a string
  - call function within a **`cout`** statement

```
string PrintName (int studentNum);
```

- function performs its own printing

Quick Note on “Print” Functions

two basic ways to handle printing:

- function returns a string
  - call function within a **`cout`** statement

```
string PrintName (int studentNum);
```

- function performs its own printing
  - call function separately from a **`cout`** statement

```
void PrintName (int studentNum);
```

Outline

- Changes for C++
  - Files & Compiling
  - Variables
  - Functions
- Input/Output in C++
  - cin/cout/cerr
  - Print Functions
  - Reading/Writing to Files
- hello_world.cpp

Reading In Files in C++

```
FILE *ifp;
```

read/write will be specified in call to **`fopen()`**

Reading In Files in C++

~~`FILE *ifp;`~~

```
ifstream inStream;
```

read specified by variable type

- **`ifstream`** for reading

Reading In Files in C++

```
FILE *ifp;
ifstream inStream;

ifp = fopen("testFile.txt", "r");
```

read is specified by “r” in call to fopen

Reading In Files in C++

```
FILE *ifp;
ifstream inStream;

ifp = fopen("testFile.txt", "r");
inStream.open("testFile.txt");
```

read is specified by declaration of inStream as a variable of type ifstream; used by open()

Reading In Files in C++

```
FILE *ifp;
ifstream inStream;

ifp = fopen("testFile.txt", "r");
inStream.open("testFile.txt");

if ( ifp == NULL ) { /* exit */ }
```

Reading In Files in C++

```
FILE *ifp;
ifstream inStream;

ifp = fopen("testFile.txt", "r");
inStream.open("testFile.txt");

if ( ifp == NULL ) { /* exit */ }
if (!inStream) { /* exit */ }
```

Reading In Files in C++

- `ifstream inStream;`
  - declare an input file variable

- `inStream.open("testFile.txt");`
  - open a file for reading

- `if (!inStream) { /* exit */ }`
  - check to make sure file was opened

Writing to Files in C++

- very similar to reading in files
- instead of type **`ifstream`**,
  use type **`ofstream`**
- everything else is the same

Writing To Files in C++

- `ofstream outStream;`
  - declare an output file variable

- `outStream.open("testFile.txt");`
  - open a file for writing

- `if (!outStream) { /* exit */ }`
  - check to make sure file was opened

Opening Files

- the **`.open()`** call for file streams takes a **`char*`** (a C-style string)

- if you are using a C++ string variable, you must give it a C-style string
- calling .c_str() will return a C-style string

```
cppString.c_str()
stream.open(cppString.c_str() );
```

Using File Streams in C++

- once file is correctly opened, use your **`ifstream`** and **`ostream`** variables the same as you would use **`cin`** and **`cout`**

```
inStm >> firstName >> lastName;

outStm << firstName << " "
       << lastName << endl;
```

Advantages of Streams

- does not use placeholders (**`%d`**, **`%s`**, etc.)
  - no placeholder type-matching errors
- can split onto multiple lines easily
- precision with printing can be easier
  - once set using **`setf()`**, the effect remains until changed with another call to **`setf()`**

Finding EOF with ifstream – Way 1

- use **>>**'s boolean return to your advantage

```
while (inStream >> x)
{
  // do stuff with x
}
```

Finding EOF with ifstream – Way 2

- use a “priming read”

```
inStream >> x;

while( !inStream.eof() )
{
   // do stuff with x

   // read in next x
   inStream >> x;
}
```

Outline

- Changes for C++
  - Files & Compiling
  - Variables
  - Functions
- Input/Output in C++
  - cin/cout/cerr
  - Print Functions
  - Reading/Writing to Files
- hello_world.cpp

hello_world.cpp

```
/* let's convert this to use
   streams and C++'s library */
#include <stdio.h>

int main()  {
  printf("Hello world!\n");
  return 0;
}
```

LIVECODING