COMP 3522

Object Oriented Programming in C++

Week 2, Day 1

Announcements

- We have a marker!
  - Eric Qiu
- He will provide feedback on marked labs
  - At the end of labs/assignments send him your first name, last name, student number, and github address
- Ask him any questions about labs he marked
- Quiz tomorrow during lecture. Before 10 minute break
  - Previous week's material
  - 10 multiple choice questions
  - The Learning Hub

Agenda

1. More File IO
2. Arrays in C++
3. Random number generation
4. Pointers, nullptr
5. References

COMP 3522

A word about implicit conversion

- **Automatically performed** by the compiler for us
- We saw this in Java when we stored a byte in a short, or an int in a long (**promotion**)
- Implicit conversions **take place when a value is copied or assigned to a compatible type**

http://en.cppreference.com/w/cpp/language/implicit_conversion

Implicit/standard conversions (fundamentals)

1. Integral promotion
    1. Widening conversion
    2. “Value-preserving”
2. Floating point conversion
    1. float to double, or double to long double are safe
    2. Narrowing is not defined if the original cannot be represented precisely
3. Conversions between integral and floating point types
    1. Truncation alert! (2.25 -> 2)

http://en.cppreference.com/w/cpp/language/implicit_conversion

Implicit/standard conversions (fundamentals)

4. **Arithmetic conversions (if one operand is a double, then…)**

```
int intValue;
double doubleValue;
double a = intValue + doubleValue
```

5. **Pointer/reference conversions (pointer/reference to derived object converted to base pointer/reference)**

http://en.cppreference.com/w/cpp/language/implicit_conversion

IO Part 2: Files

- Defined in the <fstream> header
  1. **ifstream** for reading from a file
  2. **ofstream** for writing to a file
  3. **fstream** for reading and writing to/from a file

- We can use <<, >>, and manipulators with file streams

Opening a file

```
#include <fstream>
fstream f{“data.txt”}; // Opens data for writing
if (!f.is_open()) { // Or if (!f) …
    cerr << “Unable to open file” << endl;
    exit(1);
}
f << “hello” << 123 << endl; // file closed
                             // automatically
```

Opening a file

```
// Open a file for reading
ifstream fin;
fin.open("helloWorld.txt");

// open a file (or create it if it doesn't exist)
// for writing
ofstream fout;
fout.open("helloWorld.txt");

// open a file for reading and writing.
fstream fs;
fs.open("helloWorld.txt");
```

How do we close a file

- Too easy for its own slide, but here we are anyway:

```
fin.close();
fout.close();
fs.close();
```

That's it!

Buffers

- Stream objects use an internal buffer
  - Filestreams use a filebuf
    http://www.cplusplus.com/reference/fstream/filebuf/
  - IO streams like cin, cout, cerr use a streambuf
    http://www.cplusplus.com/reference/streambuf/streambuf/
  - Stringstreams use a stringbuf
    http://www.cplusplus.com/reference/sstream/stringbuf/

- We will rarely need to manage the internal buffer directly, but it is a good idea to understand the concepts

Opening streams

- When we open a stream we can specify the “mode”
- This is similar to C
- The mode type is std::ios_base::openmode
  1. ios_base::in (**input**) Allow input operations on the stream.
  2. ios_base::out (**output**) Allow output operations on the stream.
  3. ios_base::app (**append**) Set the stream's position indicator to the end of the stream before each output operation.

More open mode flags

- More modetypes
  4. ios_base::binary (**binary**) Open in binary mode when file contains binary data.
  5. ios_base::trunc (**trunc**ate) Discard the contents of the stream when opening
  6. ios_base::ate (**at end**) Set the stream's position indicator to the end of the stream on opening.

Combine modes with bitwise OR (|)

```
ifstream f1{”data”, ios_base::in |
                    ios_base::binary};

ofstream f2{“dest”, ios_base::out |
                    ios_base::app};
```

So how do we read/write char by char?

Similar to C:

1. Use **std::basic_istream::get** to acquire the char
2. Use **std::basic_istream::put** to place the char

```
char c;
while ((c = in.get()) != EOF)
{
    // Do something
}
```

fileIO.cpp

Coding challenge

Do you remember what went into **copying a text file** in Java?

Lots of stuff, but it's fairly easy, right?

It's easier in C++ (I can't believe I just admitted this).

Try it!

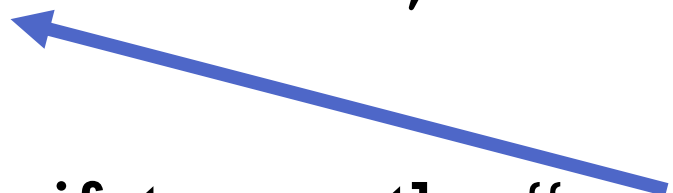
copyFile.cpp

Seeking within a file

- We now know how to read/write using file streams
- But now we need a way to navigate the file
  - We don't always want to read from the beginning of file
  - Maybe we want to jump to the middle, or end

- Imagine a text file that contains the following:

```
Hi class, here is some text
```



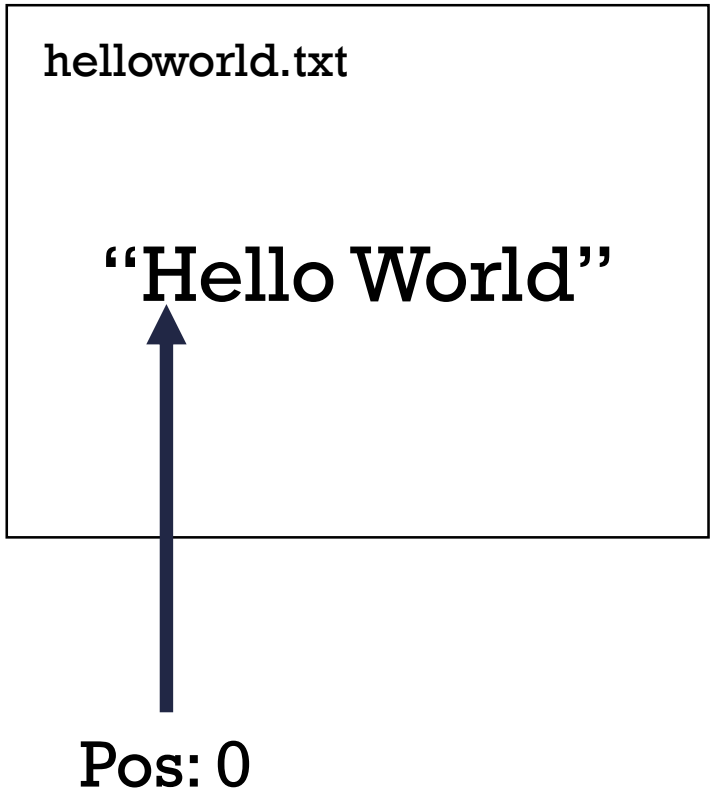
- With ifstream the "cursor" is placed at the beginning, position 0

Seeking within a file

These GET the position of the current character in the stream:

```
streampos std::ostream::tellp() // Returns the position of the current character in the output stream
streampos std::istream::tellg() // Returns the position of the current character in the input stream
```

Seeking within a file



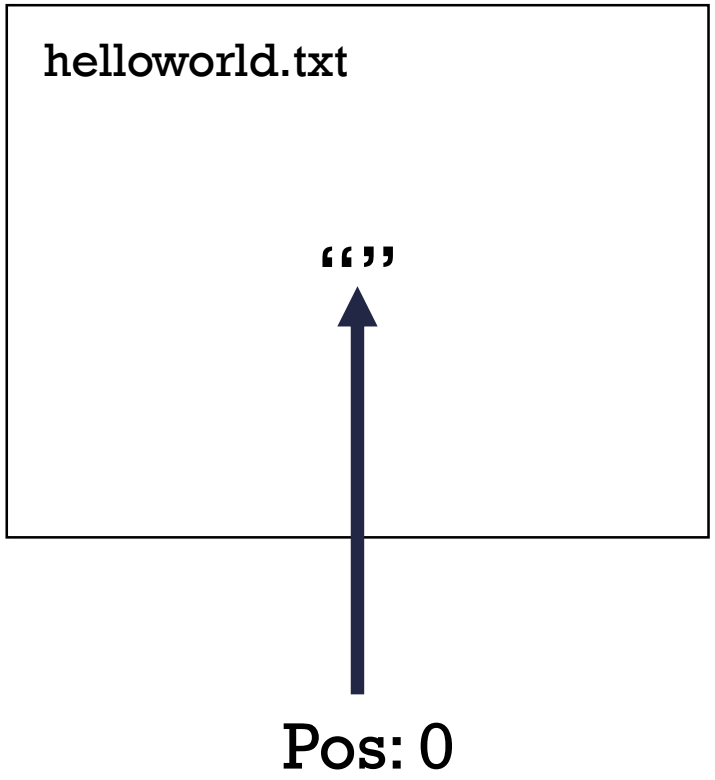

```
ifstream myFile(“helloWorld.txt");

cout << myFile.tellg() << endl;
```

Output is 0
tellg() shows current position of “cursor”

Note we use tellg for **ifstream**

Seeking within a file



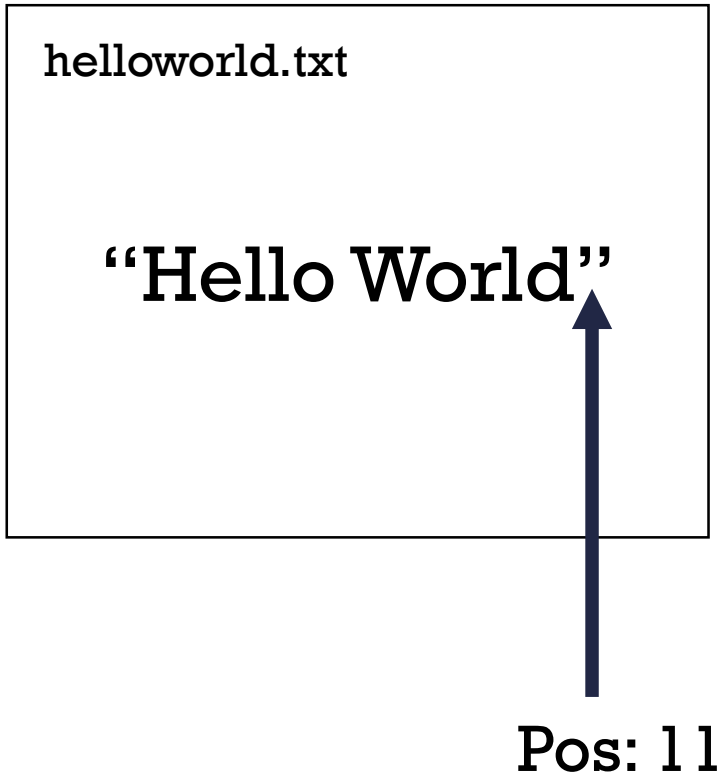

```
ofstream myFile(“helloWorld.txt“);

cout << myFile.tellp() << endl;
```

Output is 0
tellg() shows current position of “cursor”

Note we use tellp for **ofstream**

Seeking within a file




```
ofstream myFile(“helloWorld.txt“,
ios::app);

cout << myFile.tellp() << endl;
```

Output is 11
tellg() shows current position of “cursor”

Note we use tellp for **ofstream**

Seeking within a file

These SET the position of the current character in the stream:

//use seekp for output streams

```
fstream& seekp(streampos)
fstream& seekp(streamoff, ios_base::seekdir)
```

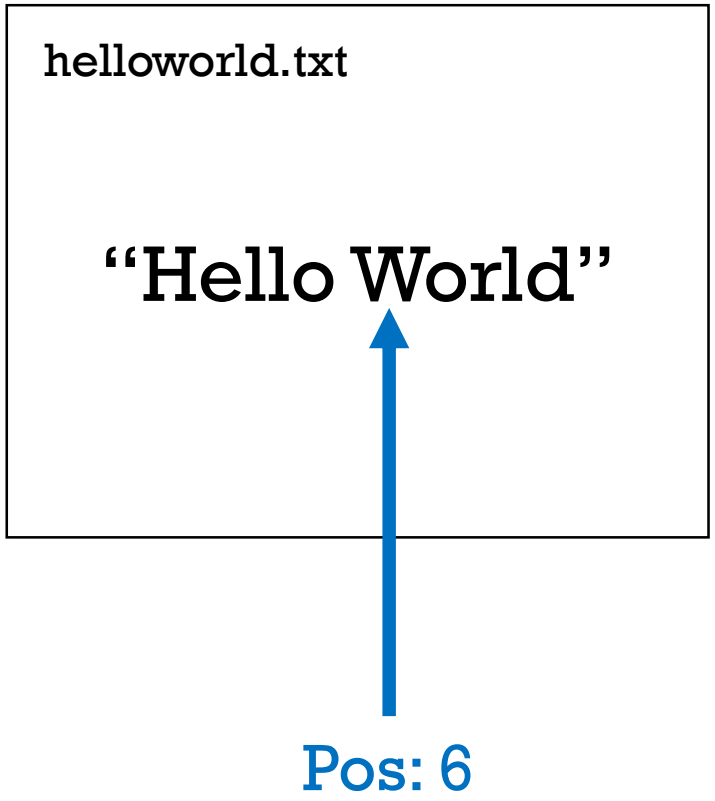
//use seekg for input streams

```
fstream& seekg(streampos)
fstream& seekg(streamoff, ios_base::seekdir)
```

Seeking within a file

- Recall C: fseek, ftell, SEEK_SET, SEEK_CUR, SEEK_END, etc.
- Similar in C++:
  - std::ios::**streampos** for storing positions
  - Following two combined to find offset relative to some position
    - std::ios::**streamoff** for storing offsets
    - std::ios_base::**seekdir** represents the seeking direction of a stream-seeking operation
      - ios::**beg** (public member of ios_base class)
      - ios::**cur** (public member of ios_base class)
      - ios::**end** (public member of ios_base class)

Seeking within a file



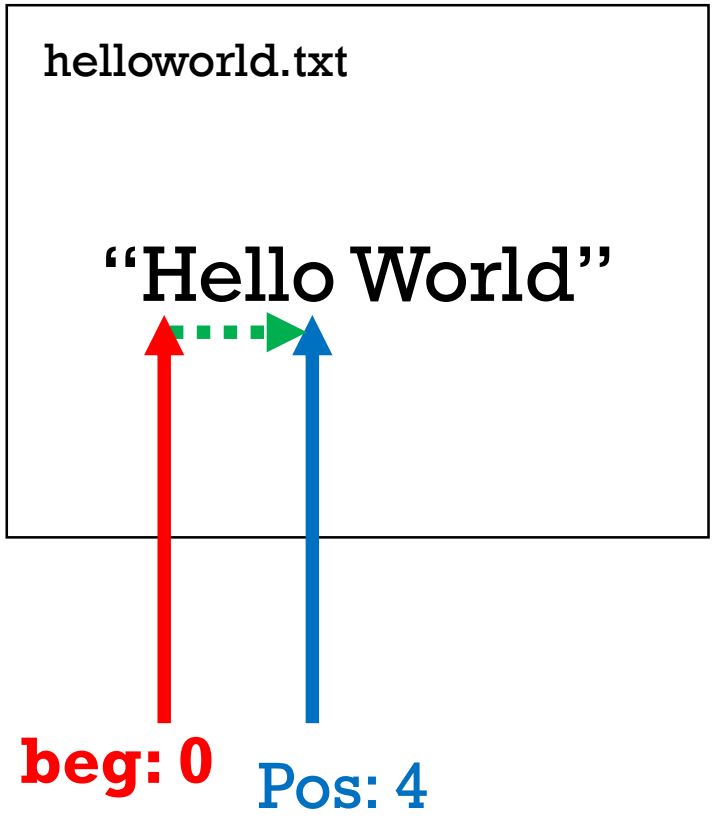

```
ofstream myFile(“helloWorld.txt");

myFile.seekp(6);
cout << myFile.tellp() << endl;
```

Sets “cursor” to absolute position 6

Output is 6

Seeking within a file



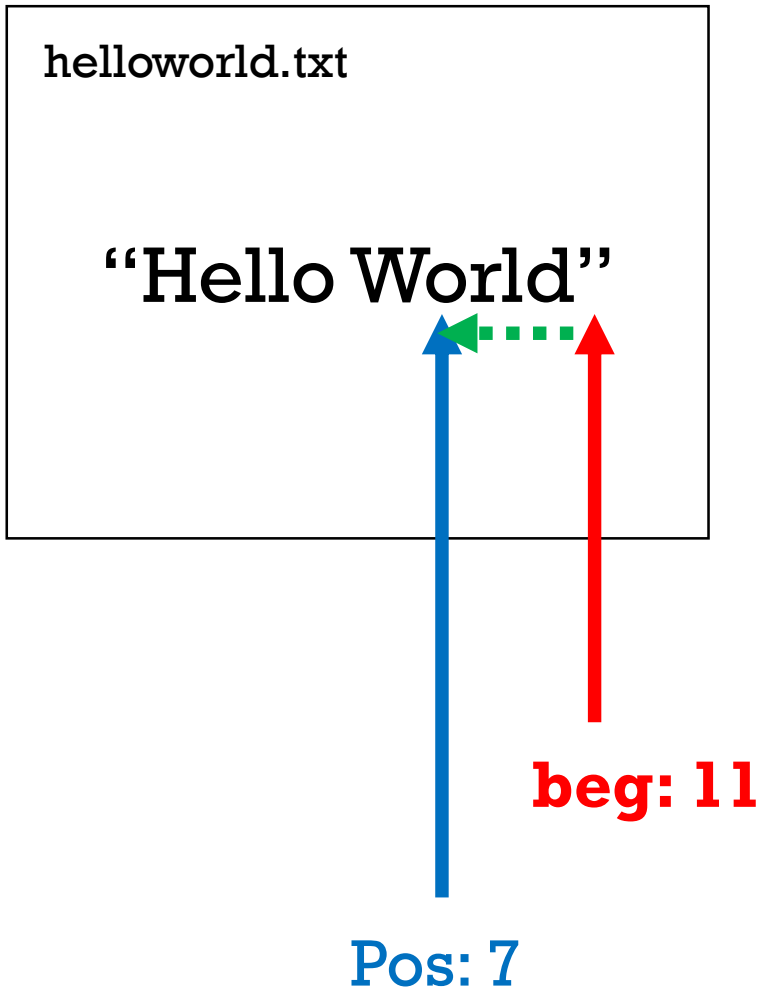

```
ofstream myFile(“helloWorld.txt");

myFile.seekp(4, ios::beg);
cout << myFile.tellp() << endl;
```

Moves “cursor” +4 positions relative to the **beg**inning

Output is 4

Seeking within a file



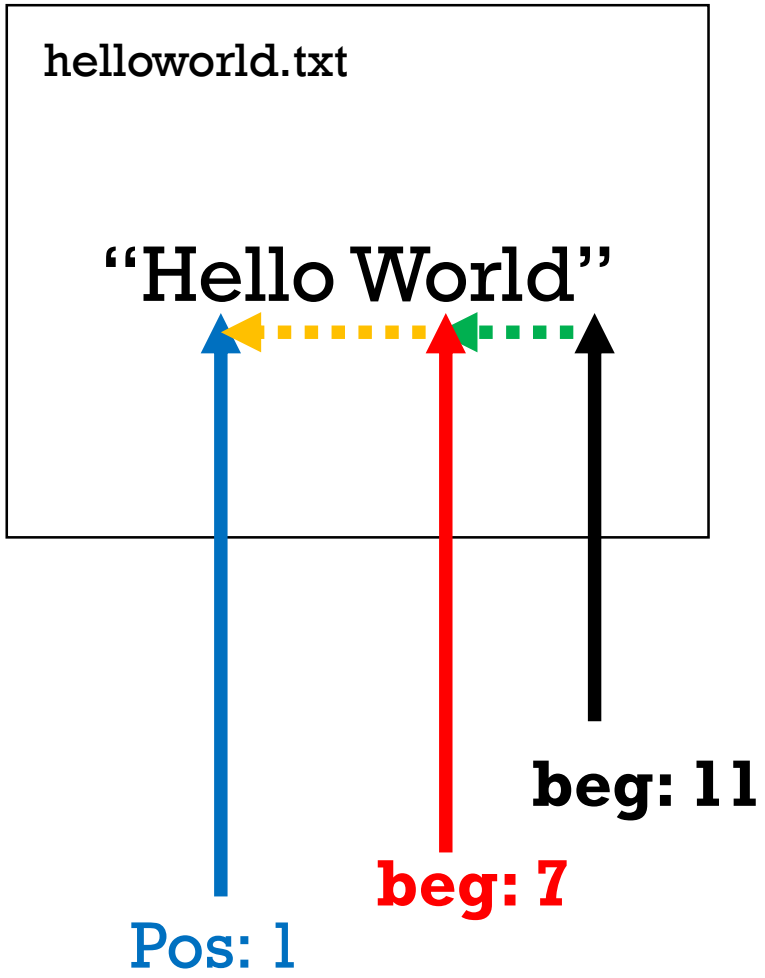

```
ofstream myFile(“helloWorld.txt");

myFile.seekp(-4, ios::end);
cout << myFile.tellp() << endl;
```

Moves “cursor” -4 positions relative to the **end**

Output is 7

Seeking within a file




```
ofstream myFile(“helloWorld.txt");

myFile.seekp(-4, ios::end);
myFile.seekp(-6, ios::cur)
cout << myFile.tellp() << endl;
```

Moves “cursor” -4 positions relative to the end.
The moves cursor -6 relative to last known “cursor” position to 1

Output is 1

NOTE
use seekp for ofstream
use seekg for ifstream
use seekp or seekg for fstream

Example code: acquiring a file's size!

```
#include <iostream>
#include <fstream>
using namespace std;
int main ()
{
   ifstream myfile{"Macbeth.txt"};
   streampos begin = myfile.tellg();
   myfile.seekg (0, ios::end);
   streampos end = myfile.tellg();
   myfile.close();
   cout << "size is: " << (end-begin) << " bytes.\n";
   return 0;
}
```

fileSize.cpp fileSeek.cpp

C++ ARRAYS

What about arrays? One slide!

```
float values[3] // array of 3 floats
char * names[32] // array of 32 pointers to
char


int scores[] = {1, 2, 3, 4};
int some_scores[8] = {1, 2, 3, 4};
    // equivalent to {1, 2, 3, 4, 0, 0, 0, 0}
```

ACTIVITY

1. Let's examine Project Gutenberg
2. Let's pick a file together
3. Count the number of occurrences of:
    1. the character Q
    2. the character q
    3. CHALLENGE: the integer 22.
4. Print the result to standard output.

ACTIVITY

Challenge: Modify your program from earlier to determine which 10 words occur most frequently in the Gutenberg file we downloaded. A word is any sequence of char delineated by whitespace. List the top 10 words and their frequency in a file called TopWords.txt

RANDOM NUMBERS

Random numbers

- There are **some fun ways to generate random numbers** in C++
- It would be helpful to review what we've seen and look at some C++ ways to make random numbers:

1. Ye olde tyme C approach

- We can generate random numbers using rand and srand from the C library:
  - srand initializes the random number generator
  - srand accepts a parameter which is a SEED (use the current time!)
  - rand returns a pseudo-random integer between 0 and RAND_MAX

```
#include <cstdlib>
#include <ctime>

srand (time(NULL));
int upperbound = 10
int my_int = rand() % upperbound;
double zero_to_one = rand() / (double) RAND_MAX;
```

1. Ye olde tyme C approach

- We can generate random numbers using rand and srand from the C library:
  - srand initializes the random number generator
  - srand accepts a parameter which is a SEED (use the current time!)
  - rand returns a pseudo-random integer between 0 and RAND_MAX

```
#include <cstdlib>
#include <ctime>

srand (time(NULL));
int random_num_1 = rand() % 100; //random range 0 to 99
int random_num_2 = rand() % 100 + 1; //random range 1 to 100
int random_num_3 = rand() % 25 + 2000; //random range 2000 to 2024
```

2. Uniform distribution of double in [a, b]

```
#include <random>
#include <ctime>

double a = 10;
double b = 100
default_random_engine generator(time(0));
uniform_real_distribution<double>
                        distribution(a, b);
double my_random = distribution(generator);
```

3. Uniform distribution of int in [a, b]

```
#include <random>

random_device rd; // a random number generator
mt19937 generator(rd()); // calls operator()
uniform_int_distribution<> distribution(a, b);
int my_int = distribution(generator);
```

Check it out! We're using a random number generator to generate a random seed for a random number generator!

random_c.cpp, random_int.cpp, random_double.cpp

How to organize our thoughts

- We have:
  - Random number generators/engines
  - Random number distributions that convert the output of random number engines into various statistical distributions
- Play with these!

Read me: http://en.cppreference.com/w/cpp/numeric/random

ACTIVITY

Create a tiny guessing game

Ask the player to guess a number between 1-10

After input, tell them if the number is too low or too high

Keep asking until they guess the correct number

Print a congratulation message when they guess the number

POINTERS, REFERENCES, AND nullptr

Call by value: will this work?

```
void swap(int arg1, int arg2)
{
     int temp{arg1};
     arg1 = arg2;
     arg2 = temp;
}

int main()
{
     int first{3512};
     int second{2526};

     swap(first, second);
     //does first = 2526 and second = 3512?
}
```

What about this?

```
void swap(int array [], int arg1, int arg2)
{
     int temp{array[arg1]};
     array[arg1] = array[arg2];
     array[arg2] = temp;
}

int main()
{
     int numbers [] = {1, 2, 3, 4};
     swap(numbers, 0, 3);
}
```

Passing pointers: what about this?

```
void swap(int* arg1, int* arg2)
{
     int temp{*arg1};
     *arg1 = *arg2;
     *arg2 = temp;
}

int main()
{
     int first{3512};
     int second{2526};

     swap(&first, &second);
}
```

swap_example.cpp

Introducing the C++ reference (&)

- An **alias** (anything done to the reference is done to the referent)
- Must be initialized when created
- Makes **pass by reference** effortless
- Used for efficiency (don't want to make a copy)

References

```
int n{123};
int& ref = n;
int m{345};
ref = m; // same as n = m
cout << n << endl; // 345
```

References as function parameters

```
void swap(int& first, int& second)
{
     int tmp{first};
     first = second;
     second = tmp;
}
…
int a{3512};
int b{2526};
swap(a, b);
```

Pointers vs references

- **Does our processor know about references?**

- **NO!**
  - Pointers and references produce the same assembly instructions
  - References are for programmers
  - References are converted to pointers when our code is compiled

References to constants

We **cannot** create a reference to a temporary value

```
int& reference{1};          // Will not compile
const int& r{1};            // OK

int n{12};
long& ref = n;              // Won't compile either! (Why not)
const long& ref = n;        // OK
```

pointers.cpp, references.cpp

Pointers and references

- Assignment to a pointer changes the pointer's value (not the pointed-to value)
- To get a pointer we need to use new or &
- To access something pointed to by a pointer (dereference), we use * or [ ]
- **We cannot make a reference refer to a different thing after initialization**
- Assignment to a reference changes the value of the object referred to (not the reference itself)
- **Assignment of references does a deep copy (assigns to the referred-to object); assignment of pointers does not**
- Beware of null pointers (assign empty pointers to nullptr as much as possible!)

IN CLASS ACTIVITY

1. Answer the questions on the References and Pointers file on The Learning Hub.
2. The Learning Hub > Content > Activities
3. Work with a partner.