COMP 3522

Object Oriented Programming in C++

Week 1, Day 1

Slides adapted from Chris Thompson

Agenda

1. Introduction
2. Toolchain
3. C++ program structure
4. Fundamental types

COMP 3522

INTRODUCTION

Me

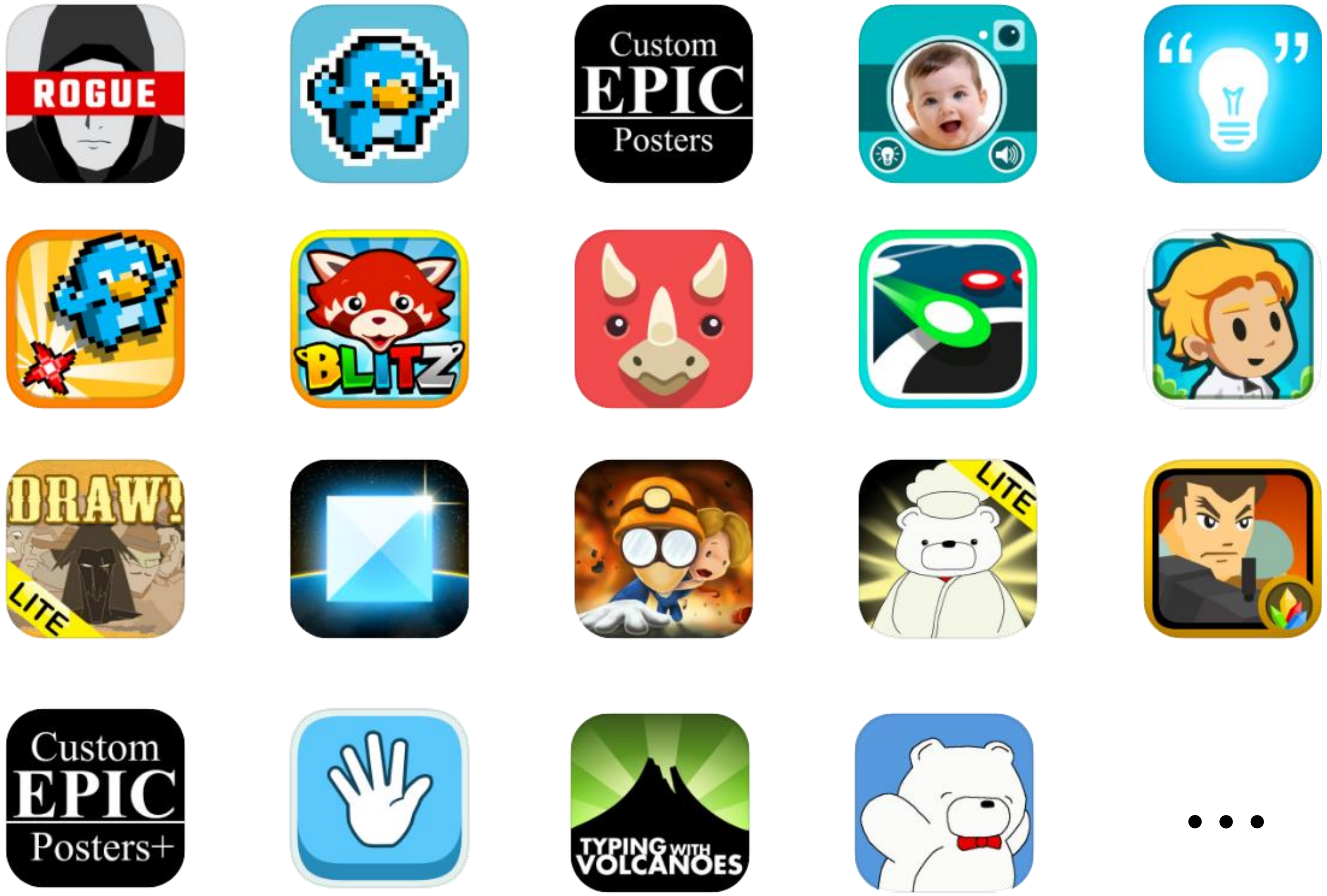
- Jeffrey Yim (call me **Jeff**)
- Email:
  - **jyim3@bcit.ca**
  - **Subject line [COMP3522]**
- Office Hours
  - SW2 - 127
  - Monday 2:30pm -5:20pm
  - Tuesday 2:30pm - 3:20pm
  - Thursday 1:30pm - 2:20pm

Me

- Education:
  - Started at BCIT! w/Pascal
  - Queen's University
  - Bachelor's/Master's Computer Science
- Interests:
  - Game development, technology
  - Unity
- Favorite language
  - C#

Industry games

iOS games


ROGUE
Custom EPIC Posters
BLITZ
DRAW!
LITE
LITE
Custom EPIC Posters+
TYPING WITH VOLCANOES

...

You – 5 mins

- Please tell us:
  1. Your name
  2. Why are you taking this course
  3. One interesting thing about you

COMP 3522

OOP: a paradigm in programming which deals with classes and objects.

A number of features of the C++ language will be covered including:

- **Inheritance**
- **Polymorphism**
- **Templates**
- **Exceptions**
- **the Standard Template Library.**

Learning Outcomes

1. Design and code basic C++ programs.
2. Understand abstract data types as represented in C++ code.
3. Design and code good C++ classes.
4. Understand and use common algorithms expressed in C++.
5. Use inheritance to capture and reuse common behavior.
6. Use polymorphism to create easily extensible systems.
7. Use templates to create reusable containers and iterators.
8. Use multiple inheritance to model complex abstractions.
9. Use exception handling to catch errors and properly release resources.
10. Use simple persistence strategies for preserving objects between program invocations and/or share objects between programs.
11. Use the standard C++ library.

Evaluation

- Your grading scheme:

• Labs	10%
• Quizzes	10%
• Assignments	20%
• Midterm	30%
• Final Exam	30%

- **<u>You must pass the final+midterm to pass the course</u>**
- A passing grade is 50.0%

Schedule

- **Lectures**
  - **Monday 11:30am – 1:20pm**
  - **Tuesday 12:30pm-2:20pm**
- Attendance is **mandatory**
- Try your best to be punctual
- Attendance is taken during labs
  - I need to report unapproved absences
  - Unapproved absence of 10% or more of the labs may result in failure or forced withdrawal from this course

Resources: Texts

1. The C++ Programming Language 4th Edition
2. The C++ Primer Plus 6th Edition

These are optional (but great to leaf through)

Lecuture Breakdown

1. Traditional lecture
2. Examine code – have small discussion
3. In-class activities
4. In-class quizzes
5. Questions always welcome
6. Give me visual feedback
   1. nodding
   2. shaking head

Resources: Online

1. http://en.cppreference.com
2. http://www.cplusplus.com/
3. https://stackoverflow.com/questions/388242/the-definitive-c-book-guide-and-list
4. https://isocpp.org/

5. https://www.tutorialspoint.com/cplusplus/
6. https://www.geeksforgeeks.org/c-plus-plus/

The Learning Hub

Will use for grades and posting lecture notes/labs/assignments

Github

Will use for submitting assignments

Academic Conduct

- Come to class
  - Quizzes during some lectures
  - We will code together
- **Turn off your ringer and put your phone away**
- Try to stay away from social media
- Be respectful and kind

Collaboration and Plagiarism

- You are encouraged to collaborate by:
  - Completing in-class exercises in pairs
  - Helping each other understand material and assignments
  - Discussing requirements and approaches
- What's not allowed:
  - Exchanging or sharing code snippets/solutions
  - Submitting someone else's work as your own
- Academic Integrity policy www.bcit.ca/files/pdf/policies/5104.pdf

TOOLCHAIN

Toolchain

1. **Communicate with Slack**
   - Best place to ask for help
   - Can have private conversations with me or with each other
   - It is where I will share news about info about the course
   - https://join.slack.com/t/comp3522-bby-fall2019/signup
2. **Submit with Student Developer Pack from Github**
   - Unlimited private repositories
   - We will use Github this term for submitting assignments and some labs
   - **https://education.github.com/pack**

Toolchain

3. **IDE (integrated development environment)**
   - Constraints:
     - FREE
     - Supports C++14 and unit testing
   - **CLion** (free for students from JetBrains)
   - Backup - Visual Studio 2017 Community or Enterprise
     - FREE for BCIT students at **https://www.bcit.ca/its/software/**
   - g++

IN CLASS ACTIVITY

1. Sign up for Slack and send me a private message telling me your full name, student number, and preferred name
   - https://join.slack.com/t/comp3522-bby-wint2019/signup

2. Apply for a free student account at Github so we can share code using version control

   https://education.github.com/pack

C++ PROGRAM STRUCTURE

Imagine C with OOP: Welcome to C++

- Created by Bjarne Stroustrup
  - Inspired in the late 70s by the Simula 67 language
  - Began as "C with Classes"
- 1983: renamed C++
- 1998: first standardization C++98
- 2003: C++03
- 2011: C++11
- 2014: C++14
- 2017: C++17…

C++

- Multi-paradigm
  - Procedural
  - Object oriented
  - Generic programming
- Also compiled
- Familiar type system (int, float, etc)
- Somewhat verbose
- Widely used
- Pass by value or pass by reference

Hello World

```cpp
#include <iostream>

int main( )
{
    std::cout << "Hello world!" << std::endl;
    return 0;
}
```

<iostream>

- This is a header file
- Note we wrap it in angle brackets and there is no file extension

- Java has an API
- C++ has a standard library*

- The <iostream> header contains some standard stream objects like **cout**

* http://en.cppreference.com/w/cpp/header

Can we access C header files in C++?

Yes!

They are included as a subset of the C++ standard library.

The names are a bit different:

math**.h** becomes <**c**math>

limits**.h** becomes <**c**limits>

stdlib**.h** becomes <**c**stdlib>

…and so on…

The insertion operator <<

- An overloaded function (we will learn how to do this later)
- We apply it to an output stream like cout
- Can be manipulated to format the output
  - Easier than printf in C
  - Much easier than NumberFormat, DecimalFormat, etc., in Java
  - We will explore manipulators in detail later

The scope operator ::

- Similar to the dot operator in C and Java

Aside: :: vs .

Q: When do we use the scope resolution operator **::** ?

A: To access **members of a namespace or class**

Q: When do we use the dot operator **.** ?

A: To access **members of an object** (an instance of a class)

std::cout

- Predefined object of type ostream in the standard C++ library
- aka the **standard output stream** (stdout in C, System.out in Java)

```
int n = 12;
cout << n;
```

This is what actually happens:

```
cout.operator<<(n);*
```

* The function header looks like this: ostream& operator<<(int);

Other useful streams

<iostream> also provides:

- std::**cin** the standard input stream
- std::**cerr** the standard output stream for errors
- std::**clog** the standard output stream for logging

std::endl

- Called an '**output manipulator**' (we will examine manipulators later this term)
- <u>Inserts a new-line and flushes the stream</u>
- IO Stream objects in C++ (cin, cout, cerr, clog) use an internal buffer of type streambuf
- Sometimes not necessary (we can just append \n to our output)

The main method

Everything starts with the main method (just like Java and C)

```
int main()
int main(int argc, char ** argv)
```

The main method must return an int (0 by default)

Preprocessor directives

- **Instructions for the preprocessor, not the compiler**
- **NOT followed by a semi-colon (ends with new line)**
- Can use to include the header file for a library (#include)
- Can use to define constants (#define)
- Can use for conditional compilation (#ifdef, #ifndef, #if, #endif, etc.)

Namespaces

- Similar to a Java package
- **Prevents name collisions**
- Functions and objects defined in the standard C++ library are in the **std** namespace

The using keyword

- Just like Java's import
- Saves typing
- We can write **using namespace std**
- If we do this, we can write
    - cout instead of std::cout
    - endl instead of std::endl

Namespace and using option 1

```
#include <iostream>

using namespace std;

int main( )
{
    cout << "Hello world!" << endl;
    return 0;
}
```

Namespace and using option 2

```
#include <iostream>

using std::cout;
using std::endl;

int main( )
{
     cout << "Hello world!" << endl;
     return 0;
}
```

Namespace and using option 3

```
#include <iostream>

int main( )
{
    using std::cout;
    using std::endl;
    cout << "Hello world!" << endl;
    return 0;
}
```

Hello World using namespace

```
#include <iostream>

using namespace std;

int main( )
{
    cout << "Hello world!" << endl;
    return 0;
}
```

LET'S TRY IT

- Show HelloWorld.cpp in CLion

FUNDAMENTAL TYPES

Memory Lane: Java is strongly-typed

- 8 primitive types
  - byte, short, int, long
  - float, double
  - boolean
  - char
- Arrays
- Classes
- Interfaces

Memory Lane: C is also strongly-typed

- integer types
  - char, short, int, long
- floating point types
  - float, double, long double
- void
- fun stuff like
  - arrays
  - structs and unions
  - pointers!

C++ is a strongly typed language, too

- Every variable has a type
- That type never changes
- Variable declarations need:
    1. Type
    2. Variable name
    3. Optional initialization

Speaking of identifiers…

- ✓ Letters
- ✓ Digits (don’t start with digit)
- ✓ Underscores

- Begin with a letter or (**rarely**) an underscore

In C++, use bool for Boolean

Can be true or false

```
void f(int a, int b)
{
    bool b1{a == b}; //what’s this???
    // do whatever f does…
}
```

* Non-zero integers convert to true, zero converts to false

So many characters!

C++ provides a variety of character types:

1. **char** is the default
2. **signed char** is like char, but guaranteed to be signed
3. **unsigned char** is like char, but guaranteed to be unsigned
4. **wchar_t** is large enough to hold the largest character set supported by the implementation's locale
5. **char16_t** is for 16-bit sets, like UTF-16
6. **char32_t** is for 32-bit sets, like UTF-32

Integer types

Four sizes:

1. **short** (int)
2. “plain” **int**
3. **long** (int)
4. **long long** (int)

Three forms:

1. “plain”
2. **signed**
3. **unsigned**

Floating point numbers

There are three floating-point types:

1. **float** (IEEE-754 **32-bit** single precision)
2. **double** (IEEE-754 **64-bit** double precision)
3. **long double** (usually **80-bit**)

Sizes of Fundamental C++ types

1 byte == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)

1 byte <= sizeof(bool) <= sizeof(long)

sizeof(char) <= sizeof(wchar_t) <= sizeof(long)

sizeof(float) <= sizeof(double) <= sizeof(long double)

sizeof(fundamental type) == sizeof(signed fundamental type) == sizeof(unsigned fundamental type)

Don’t forget size_t

- **size_t theSize = sizeof(XXX)**
- Implementation-defined
- Unsigned integer
- Good to use when we want to store the size of an object

Sizes and ranges

- Use the **sizeof()** operator to find out a variable or type's size

- We can use constants in the **<limits>** header and the **<climits>** header to find out the maximum and minimum values http://en.cppreference.com/w/cpp/header/limits

```
cout << "Max char: " << CHAR_MAX << endl;
```

Sizes and ranges

- We can also use a class template (like a generic) declared in <limits> called **std::numeric_limits**: http://en.cppreference.com/w/cpp/types/numeric_limits

```
cout << “Max value of int: “
     << std::numeric_limits<int>::max()
     << endl;
```

Initializing variables in C++

There are 3 ways to initialize variables in C++:

1. C-like initialization
   int x = 0; // assignment operator
2. Constructor initialization (C++)
   int x(0); // parentheses
3. **Uniform initialization (C++11)**
   int x {0}; // curly braces

Why should we prefer uniform initialization?

It **prohibits implicit narrowing conversion** among built-in types

```
double x, y, z;
int sum = x + y + z; // ok (value of expression truncated to int)
int sum(x + y + z);  // same

int sum{x + y + z}; // ERROR!  This won’t work.  We’re happy!
```

Now that we know about Uniform initialization

```
void f(int a, int b)
{
    bool b1{a == b}; //what’s this???
    // do whatever f does…
}
```

Summary - Fundamental Types in C++

- Character types like **char**, char16_t, char32_t, wchar_t
- Signed integer types like signed char, short, **int**, long, long long
- Unsigned integer types like (unsigned) char, short, int, long, long long
- Floating-point types like float, **double**, long double
- Boolean (hooray!) called **bool**
- void

HOME ACTIVITY

1. Write a little program to help fill in the blanks in the chart on the next page.
2. In Java and C, we often have to CAST values to store them in variables of a different type. Is the same true in C++? Prove it!

Type	Size (B)	Minimum Value	Maximum Value
bool	1		
short int	2		
unsigned short int	2		
int	4		
unsigned int	4		
long int	4		
unsigned long int	4		
long long int	8		
unsigned long long int	8		
float	4		
double	8		
long double	8		
char	1		
unsigned char	1		

Hint

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Size of char : " << sizeof(char)
         << endl;
    return 0;
}
```

Hint 2 a

- Examine std::numeric_limits<XXX>::min() where XXX is a float
- Is that a **negative** number?

Hint 2 b
min() v lowest() for floating point numbers

- min() is the tiniest value that can be represented
- lowest() is the least value, i.e., no other value lies to the left of this value on the number line