COMP322 - Introduction to C++

Lecture 01 - Introduction

School of Computer Science

8 January 2013

What this course is

- Crash course in C++
- Only 13 lectures
- Single-credit course
- As the lectures only take up 1 hour per week, it will be your responsibility to read any assigned readings.
- Course material is partly up to you

Goals of Course

- Understand basics of OOP
- Crash course in some of the tricks of C++
- Have a decent sense of what tricks are good and what tricks are just confusing!

What this course is not

- An introduction to programming course
- A full OOP course
- A gentle tour of C and Java syntax

Prerequisites and Assumptions

- Assumes you have taken COMP206 OR COMP 202 OR COMP 250 OR COMP 208.
- Assumes you are comfortable in C programming language.
- If you are not comfortable in C but know Java, it is probably OK. You will, however, find some concepts you need to catch up on and some concepts you already know though.
- See me if you have any concerns.

Course facts

- Course web page:
  `http://www.cs.mcgill.ca/∼dpomer/comp322/winter2013`
  (if copy/paste of above link fails, try typing manually....for some reason the ∼ has an odd font)
- Office hours: Tuesdays 13:15-14:15, Thursdays 14:15-15:15 (flexible depending on necessity)
- Academic Integrity: See
  `http://www.mcgill.ca/integrity`

Assessment

- Two short quizzes, 20% each
  - Short-answer, multiple-choice, true/false
  - Given in class
- Four homework assignments, 20% each
  - One or more short programming problems
  - 3 weeks per assignment
  - 10% per day late penalty
  - Use GNU C++ ("g++")
  - Comments and style will be counted, in addition to correctness
- Final grade will be the sum of the best 5 scores, *provided* the work does not violate academic integrity standards!

A little about your instructor

- Dan Pomerantz, `dpomer@cs.mcgill.ca`, Course Lecturer
- Office: McConnell 306
- MSc. from McGill. Worked on recommender systems with Greg Dudek
- `http://www.recommendz.com`
- Afterwards worked with Bing Shopping search engine on extracting information from webpages.
- Avid New York Rangers fan

A little about C++

- Begun in 1979 by Bjarne Stroustrup
- Originally called “C with Classes”
- First used outside Bell Labs in the mid-80’s
- ANSI/ISO standard (ISO/IEC 14882:1998)
- Important ancestor of Java

Design principles

- Compiles to machine (binary) code
- Compile-time type checking
- Flexible programming styles
- Low runtime overhead
- Minimal development environment
- Mostly compatible with C

Differences from C

- Classes
- Overloading
- Templates
- Exceptions
- Namespaces

Differences from Java

- Compiles to machine code
- Multiple inheritance
- Pointers and references (exist in Java but very different)
- No garbage collection

Pros and cons

- Like C, C++ is useful for systems programming
- Commercially important!
- VERY powerful!
- Can seem complex and difficult
- Allows serious errors and security problems
- Not quite as standard as either C or Java
- Lots of “missing features”

C++ Standard Library

- Includes most of the C Standard Library
- Derived from Standard Template Library (STL)
- Data types: Strings, complex numbers, etc.
- Containers: Lists, sets, queues, stacks, etc.
- Algorithms: Sorting and searching

Topics in this course

- Pointers, references, memory management.
- Standard C++library
- Classes and object oriented programming
- Inheritance
- Templates
- Basics of exceptions
- ???

Please contact Dan if you have particular requests and, if possible, he'll try to include a bit on it.

C++ example - hello.cpp

Now let's look at our first C++program.

What do we expect this program should do?

C++ example - hello.cpp (what else?)

```
#include <iostream>
int main()
{
  std::cout << "Hello, world!\n";
  return 0;  // Return code for success
}
```

As is standard tradition, we have to start with hello world!

How to run this program

- Save the file as hello.cpp (or anything else if you prefer)
- Open up a command prompt and change to the directory
- To compile the program, type `g++ -Wall hello.cpp`. (`-Wall` is optional but recommended)
- The above will produce a file called a.out which you can run.
- If you want to compile to a different file type `g++ -Wall -o desiredExeName hello.cpp`
- To run the program type `./a.out` (on Windows you can just type the name of the exe) or `./desiredExeName`

C++ example - hello.cpp

Now let's analyze this program a bit.

C++ example - hello.cpp

```
#include <iostream>
int main()
{
  std::cout << "Hello, world!\n";
  return 0;  // Return code for success
}
```

- “#include” is a preprocessor directive
  - Preprocessor runs before the compiler
  - The entire file “iostream” is incorporated
  - No semicolon used in preprocessor statements
  - Incorporates part of standard library
  - A bit different than a Java import statement

C++ example - hello.cpp

```
#include <iostream>
int main()
{
  std::cout << "Hello, world!\n";
  return 0;   // Return code for success
}
```

- "main()" is a special function
  - Control starts with this function
  - It must be a global function returning `int`
  - Must be defined only once per project
  - Is *not* part of any class

C++ example - hello.cpp

```
#include <iostream>
int main()
{
  std::cout << "Hello, world!\n";
  return 0;  // Return code for success
}
```

- `std::cout` refers to a global object
  - It is an object of the class `ostream`
  - It is similar to the `stdout` global from C
  - The << operator writes the object
  - The '::' is the scope operator

C++ example - hello.cpp

```
#include <iostream>
int main()
{
  std::cout << "Hello, world!\n";
  return 0;  // Return code for success
}
```

- `return` specifies value of function `main()`
  - Takes an (optional) value
  - The number zero is an integer constant
  - In this case, zero indicates success
  - Returns control to calling function

C++ example - Compiling and running

```
$ g++ -Wall -o hw hello.cpp
$ ./hw
Hello, world!
$
```

The flag `-Wall` means “Warning all” This means the compiler will check for additional “questionable” things such as an unused variable. Note that it is a *warning* and NOT an error. It is highly highly highly highly highly recommended that you use this flag!

C++ basics

- Statements terminated with semicolon
- Comments either between `/* .. */` or after `//`
- Basic constants and types largely borrowed from C
- Most operators identical to those in C
- Parentheses are used to group expressions: `a * (b + c)`
- All identifiers must be declared before use, e.g.
  `int inch; float sum = 0.0;`

C++ basics - Basic types

The sizes and specific range values are typical for 32-bit systems.

Type	Bytes	Min	Max
bool	1	false	true
signed char	1	SCHAR_MIN (-128)	SCHAR_MAX (127)
unsigned char		1	0
char	1	CHAR_MIN	CHAR_MAX
short [int]	2	SHRT_MIN (-32768)	SHRT_MAX (32767)
unsigned short [int]	2	0	USHRT_MAX (65535)
int	4	INT_MIN	INT_MAX
unsigned [int]	4	0	UINT_MAX
long [int]	4	LONG_MIN	LONG_MAX
unsigned long [int]	4	0	ULONG_MAX
float	4	-FLT_MAX	+FLT_MAX
double	8	-DBL_MAX	+DBL_MAX
long double	8	-LDBL_MAX	+LDBL_MAX

C++ basics - Arithmetic operators

Where possible, C++ will automatically convert among the basic types.

```
+       // Addition
-       // Subtraction
*       // Multiplication
/       // Division
%       // Integer remainder
```

Another important operator is the assignment operator:

```
=       // Assignment
```

C++ basics - Comparison operators

The result of a comparison operator is always a value of type 'bool':

```
==      // equal
!=      // not equal
>       // greater than
<       // less than
>=      // greater than or equal
<=      // less than or equal
```

C++ basics - Logical operators

The logical `&&` and `||` operators use short-circuit evaluation. They execute the right hand argument only if necessary to determine the overall value.

```
&&    // logical and
||    // logical or
!     // logical negation
```

C++ basics - Bitwise operators

These operators support logical operations on bits. For example,

```
int x = 0x1001 ^ 0x2001;
std::cout << std::hex << x << std::endl;
```

would print 3000.

```
&     // bitwise and
|     // bitwise or
^     // bitwise exclusive or
~     // bitwise complement
<<    // left shift
>>    // right shift
```

C++ basics - if statement

```
// Simplest form
if (response == 'y') return true;

// Less simple
if (result > 0.0) {
  x = 1.0 / result;
  y += x;
}
else {
  std::cout << "Division by zero!";
}
```

C++ basics - `switch` statement

```
int response;

std::cin >> response; // Get input

switch (response) {
case 'y':
   return true;
case 'n':
   return false;
case 'q':
   exit(0);
default:
   std::cout << "I didn't get that, sorry\n";
   break;
}
```

C++ basics - `while` statement

```
float array[10];
int i;

i = 0;
while (i < 10) {
    array[i] = 0;
    i++;
}
```

C++ basics - `for` statement

Typically a shorthand for common forms of the `while` statement.

```
float array[10];

for (int i = 0; i < 10; i++) {
   array[i] = 0;
}
```

C++ basics - do while statement

```
int response;
do {
  std::cin >> response;
  processCommand(response)
} while (response != 'q');
```

C++ basics - Identifier scope

```
int v = 1;     // Global scope

int main()
{
  int c = 5;  // Local scope

  // Declare 'i' in statement scope
  for (int i = 0; i < c; i++) {
    // do something
  }
  // 'i' is now undefined
  c = c + v;
}
```

C++ basics - Functions

```
/* Calculate the mean of an array */
double mean(double data[], int n)
{
  double sum = 0.0;  // Initialization
  if (n != 0) return 0.0;
  for (int i = 0; i < n; i++)
    sum += data[i];
  return sum / n;
}

/* Impractical recursive factorial */
long factorial(long t)
{
  if (t <= 1) return 1;
  return t * factorial(t - 1);
}
```

Preprocessor

The C++ preprocessor is inherited from C. It runs before the compiler, processing its directives and outputting a modified version of the input.

Any statement starting with # is a preprocessor command. We will see some uses for this throughout the term.

```
#define   #include
#ifdef    #ifndef
#if       #elif
#else     #endif
#line     #undef
#error    #pragma
```