COMP6771
Advanced C++ Programming

Week 1
Part 1: Course Overview

2017

www.cse.unsw.edu.au/~cs6771

About the Course

COMP6771 is a programming course, teaching practical aspects of intermediate/advanced C++ programming.

- Intensive programming components
- Requires a considerable amount of time and hard work
- Relatively fast-paced teaching
- Not an introductory programming course
- Not an OO design course
- Not a course on C
- Not an easy course

After Week 1, you are expected to read the first 300+ pages from the textbook.

Prerequisites

Formally, you must have passed the following:

- Undergraduate: COMP2911
- Postgraduate: COMP9024

Informally, you should posses/exhibit the following:

- good knowledge of C (and some experience with Java/Python/C#)
- you are comfortable with pointers, arrays, linked lists, trees, dynamic memory allocation, hashing and function pointers.
- an understanding of key OO concepts (such as classes, encapsulation, inheritance and polymorphism)
- ability to understand and solve problems
- ability to code, code, code

Course Structure

Three hours of lectures per week

- Covers topics from the textbook

One hour tutorial per week (starting week 2)

- Reinforces material from lectures and readings
- Practises your problem-solving skills

Five programming assignments

- Develop your programming skills

Three-hour final exam

Course Staff and Administration

Lecturer in Charge:

Alex Bokani:	abokani@cse.unsw.edu.au

Tutors:

Matthew Stark:	m.stark@unsw.edu.au
Peter Kydd:	p.kydd@unsw.edu.au
Donny Yang:	donny.yong@unsw.edu.au
Henry Zhang:	henry.zhang@unsw.edu.au
Joshua Pratt:	joshua.pratt@unsw.edu.au

Important messages will be displayed on the course website and urgent messages also sent to you by e-mail.

Check the course website and your email regularly.

Course Outline

Using Abstractions (about 1/4 of the lectures):

- Course Introduction, C++ Basics
- STL Containers, Iterators and Generic Algorithms

Building Abstractions:

- Classes: `this`, Scope, Constructors, Friends, Statics
- Copy Control, Overloaded Operators, Conversions
- Inheritance, Dynamic Binding, Object Model, Copy Control
- Function and Class Templates
- Memory Management, Exception Handling
- Templates and Iterators
- Multithreading
- Metraprogramming and Multiple Inheritance

Programming Assignments

The total weight for assignments is 45%.

Five individual programming assignments

Only a small number of test cases may be given for ensuring correct input/output format

$\Longrightarrow$ Testing is part of the assignments!

Assignment	Tentative Release Date
1	Week 1
2	Week 3
3	Week 7
4	Week 9
5	Week 11

Assignment 1 will be released this week.

Programming Assignments

Plagiarism will not be tolerated and is serious academic misconduct.

- Plagiarism is more than just copying code.
- Sharing (including uploding to github), buying, or selling code from peers or others are all forms of plagiarism.

The course outline has further information about plagiarism and late submissions

Ensure your code compiles on the school computers before the assignment deadline.

Assignment Compilation

We use GCC 4.9 on school computers, generally with the following flags:

```
g++ -std=c++14 -Wall -Werror -O2 -o assignment
assignment.cpp
```

g++ Compiler Upgrade

g++ 4.9.2

The g++ compiler on the school computers (used for assignment marking) is version 4.9.2
You should test your assignment code against this version on the school computers before the assignment submission deadline.

To install this version of the compiler on Ubuntu 14.04:

sudo add-apt-repository ppa:ubuntu-toolchain-r/test

sudo apt-get update

sudo apt-get install g++-4.9

you can then compile using the command:

```
g++-4.9 -std=c++14 file.cpp
```

Programming Assignments

Style Rules:

- Macros should never be required in C++.
- Variables should be defined close to their use.
- Don't use `malloc`/`realloc`, use `new` (or smart pointers)
- Minimise use of `void*`, pointer arithmetic, union and c-style casts
- Minimise the use of C-style arrays and strings, use `vector`/array (from C++11) and `string` instead

Final Exam

The total weight for final exam is 55%.

Three-hour and closed-book

All materials covered in this course are examinable

- lectures, set readings, tutorials and assignments

The final exam is a hurdle requirement

You must pass the final exam *and* get an overall mark of 50% in order to pass this course.

Course Resources

Course website:

http://www.cse.unsw.edu.au/~cs6771

- Provides links to all material related to this course.

Course forum (i.e. MessageBoard from the course website):

- *Always use it as the first stop for help!*

For lecture and administration questions email:
abokani@cse.unsw.edu.au

For tutorial and programming questions email your tutor.

Course textbook

C++ Primer, 5th Edition, 2013.
Stanley B. Lippman, Jose Lajoie, Barbara E. Moo.
ISBN-10: 0321714113. ISBN-13: 978-0321714114.

The Optiver COMP6771 Prize

http://www.optiver.com/sydney/

Awarded to the undergraduate student with the highest mark

Questions?

Read the course outline from the course homepage
http://www.cse.unsw.edu.au/~cs6771/

Next: Introduction to C++

What is C++

C++ is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language.

Invented by Bjarne Stroustrup in the early 80's

Originally called "C with classes"

C++ maintains a high degree of compatibility with C
http://stackoverflow.com/questions/1201593/c-subset-of-c-where-not-examples

Borrowed the class concept from Simula67

Some features inspired by Algol68 and also Ada, Clu and ML

- Almost a superset of C (closer to the machine)
- Functions/variables can be outside a class (unlike Java)

Greatly influenced newer languages, such as Java and C#

C++11 and C++14

C++14 is the most recent iteration of the C++ programming language.

- C++11 contained major additions to the core language:
  - *uniform initialisation; rvalue references; move semantics; automatic type deduction; new smarter pointer classes;*
  - Extended the C++ standard library
- The textbook is up to date with the C++11 standard
- C++14 included several further additions to the language and standard library.
- Course assignments will be compiled against the C++14 standard

C++ 17 vs C++ 14

https://isocpp.org/files/papers/p0636r0.html

C++ Supports Multi-Paradigm Programming

Procedure-based programming

> Decide which procedures you want; use the best algorithms you can find.

Object-based programming (i.e., data abstraction, structs)

> Decide which types you want; provide a full set of operations for each type.

Object-oriented programming (i.e, classes)

> Decide which classes you want; provide a full set of operations for each class; make commonality explicit by using inheritance.

Generic programming ($\Longrightarrow$ STL)

> Decide which algorithms you want; parameterise them so that they work for a variety of suitable types and data structures.

Key Features of C++

Basic types, operators, expressions, statements

Pointers, references and arrays

Modular programming, namespaces, separate compilation

Complex exception handling

Explicit memory management

Data abstraction, user-defined types, polymorphic types

Classes and objects, simple and multiple inheritance

Templates, generic programming and algorithms

The Standard Library

C++ Standard Library

The Standard Template Library (STL) is a part of the C++ standard library

Some others are:

- The C standard library
- Language support: `<exception>`, `<ctime>`, ...
- Strings: `<string>`, `<cstring>`, ...
- Numerics: `<cmath>`, `<complex>`, ...
- I/O: `<iostream>`, `<fstream>`, `<cstdio>`, ...
- Exception classes
- Smart pointers
- Classes for internationalisation support

C++ STL

A generic (or reusable) library for managing collections of data with modern and efficient algorithms

- Containers: vector, list, stack, ...
- Algorithms, find, sort, copy, ...
- Iterators are the glue between the two!

All components of the STL are templates

For efficiency reasons, STL is not object-oriented:

- Makes little use of inheritance, and
- Makes no use of virtual functions

Hello World

```
1 // helloworld.cpp
2 #include <iostream>
3
4 int main() {
5    std::cout << "Hello, world!" << std::endl;
6    return 0;
7 }
```

You may use Eclipse IDE (or any other IDEs) as the development environment, however we will not provide support and your code must compile off the command line on the CSE systems.

Source code for many (but not all) examples from the slides will be on the course website.

Hello World

Although simple, the program contains:

- A comment
- A `#include` directive
- Standard library elements
- The output stream, `cout`
- The stream insertion operator << (operator overloading)
- The stream manipulator `endl`
- The `std` namespace
- The `return` statement
- Semicolons, braces and string literals

Compiling

The Compiler and Optimisation Flags

We use GCC (GNU Compiler Collection) on school computers:

```
g++ -std=c++14 -Wall -Werror -O2 -o helloworld
helloworld.cpp
```

- -Wall: enable all warnings
- -Werror: Make all warnings into errors

The compilation process:

- Preprocessing - handles meta-information (e.g., #include)
- Compiling - translates C++ into machine code objects
- Linking - merges objects into a single application

C++ is Complex

Backwards compatibility with C (baggage?)

- Complex rules for name resolution
- Pointers and casting $\Longrightarrow$ performance + bugs

Emphasis of performance

- Objects allocated on stack and heap
- Explicit memory management
- Pointer arithmetic

User-defined types behave similarly as built-in ones

- Operator overloading
- Type conversions

Drawbacks:

Cryptic error messages

"Trusts" the programmer not to violate language rules

C++ Style

Place declarations in .h header files

Place definitions in .cpp source files

Some popular style guides:

- Stroustrup's: http://www.stroustrup.com/bs_faq2.html
- Google: http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml
- GCC: http://gcc.gnu.org/wiki/CppConventions

Code and comment consistency matters more than any particular style

C++ vs C

- *Good* C programs tend to be C++ programs
- C++ is a lot richer than C, so the converse is not true

Hints for C programmers:

- Macros are almost never required in C++, use `const`, `enum`, `inline`, `template`, `namespace`
- Variables should be defined close to their use
- Don't use `malloc`/`realloc`, try `new` or use smart pointers
- Minimise use of `void*`, pointer arithmetic, union and c-style casts; good program design can often avoid them (unless you are developing low-level libraries)
- Minimise the use of C-style arrays and strings, use `vector`/array (from C++11) and `string` instead

Strategies for Leaning C++

Focus on concepts and programming techniques

- Don't get lost in language features

C++ supports many different programming paradigms

Learn C++ gradually

- Don't have to know every detail of C++ to write a good C++ program

One Further Advice: Understand types, pointers, recursion and compilers!

Readings

Chapters 1 and 2

Some optional readings:
http://www.codeproject.com/Articles/570638/Ten-Cplusplus11-Features-Every-Cplusplus-Developer

http://www.cprogramming.com/c++11/what-is-c++0x.html

Header files: http://www.gamedev.net/page/resources/_/technical/general-programming/organizing-code-files-in-c-and-c-r3173

Next: Introduction to C++ cont.