NOTEWORTHY

Tuesday, 27 June 2017
3:14 pm

Probably a good idea to add to your cheat sheet

SOLID

- Adapter is good for DIP (dependency inversion)
- Decorator is good for OCP (open close)
- All wrappers conform to SRP (single responsibility), provide a simple interface

Container Constructors

`C<T> c`	Create an empty container `c` of elements of type `T`. `C` is a container, e.g., vector, list, deque.
`C<T> c(o)`	calls the copy constructor of `c`
`C<T> c(begin,end)`	`c` is created by providing a range of *iterators* from `begin` to `end`, *not* including `end`.
`C<T> c(n, init)`	Create container `c` with `n` copies of `init`
`C<T> c(n)`	Create `c` as `n` "value-initialised" items.

Course

Monday, 27 February 2017
11:25 pm

Object Oriented Design

A/Prof. Bernhard Scholz

Room: Office 411
School of IT Building, J12
Email: Bernhard.Scholz@sydney.edu.au
Consultation Hour: 4pm to 6pm Mondays in SIT Room 411

Textbook
Design Patterns by Gamma, Hein, Johnson, and Vlissides
C++ Primer 5ed by Lippman, Lajoie, and Moo

Lab work
http://sydney.edu.au/engineering/it/~info3220

Tools
C++, Qt Creator, Git

1 Introduction to C++

Thursday, 9 March 2017
9:56 am

Why Object Oriented design?

- Abstraction
- Polymorphism
- Inheritance

C++ extends C

- C++ is a superset of C
- Almost all code in C can be compiled in C++
- Not all code in C++ can be compiled in C

C++

- Compiled language
- Variables in C++ should be initialized, else they are undefined
- No garbage collection: new and delete
- Extension: ".cc", ".cp", ".cpp"
- Can inherit from multiple classes (java cannot)

Streams

- Used for printing to console or file
- << is a stream operator

Namespaces are sets

- Used to prevent name / symbol clashes
- Can specify to use a default namespace at the beginning of file
- Can use multiple namespaces in same piece of code
- Namespaces can be nested

```
#include <iostream>

using namespace std;

int main() { return 0; }
```

Use header files for declaration (void of implementation)

- Split code into modules
- Can be compiled separately (recall make file)
- #include "example.h" looks for file in current path
- Be careful to include header files only once.
    - Use preprocessor command #ifndef and #endif
    - OR, new in C++
    - #pragma ONCE

Macros are evil

- You don’t need them with what C++ offers

Method declarations go in header file
Implementation of methods go in CPP file

Lab 02

Monday, 13 March 2017
2:17 pm

This course is 50% design patterns and 50% C++

Header guards:

```
#ifndef __FOO_H__

#define __FOO_H__

…

#endif
```

Namespaces

```
namespace example {

}

example::foo // look for foo inside the container 'example'

using namespace example; // should never appear in a header file
```

Initializer list:

When implementing constructors,
Initialize values in the order they are declared

Pure virtual methods:

They combine the ideology of interfaces and abstract classes in Java.

```
virtual int getSpeed(); // no requirements to initialize

virtual int getSpeed() = 0; // method must exist, must initialize
```

2 More C++

Thursday, 16 March 2017
10:09 am

Pointers

- Reference (&) and dereference (*) operations
- Make sure they are initialized, otherwise UNDEFINED

```
std:string *p1, *p2;
```

- Why pointers? Passing around an address is faster than passing around a whole object

Default arguments

- The constructor can have default arguments
- Be careful not to make constructors ambiguous

```
Ball() { m_radius = 0; m_rgb = 0; }

Ball(double radius = 1, int colour = 1) {}
```

Initializer lists

- A value is assigned when the variable is created in memory
- Should be in same order as variables in class declaration

This:

```
1  class Ball
2  {
3  public:
4     Ball(double radius, int colour)
5     {
6        m_radius = radius;
7        m_rgb = colour;
8     }
9  private:
10    const double m_radius;
11    const int m_rgb;
12 };
```

won't compile. I'm trying to "initialise" the const value twice!

But this:

```
1  class Ball
2  {
3  public:
4     Ball(double radius, int colour)
5           : m_radius(radius), m_rgb(colour)
6     {
7
8     }
9  private:
10    const double m_radius;
11    const int m_rgb;
12 };
```

will. ☺
The initialisation is only attempted once, as it should be.

Const

- Const variables cannot change their value during their lifetime
- The left code won't work because the variables have been created before you move into the body
- Const variables are a different type
  - Const int is not the same as an int
  - You can move from non-const to const
  - You cannot go the other way
- Const methods