Advanced Topics

COSC1076
Semester 1 2019
Week 11

RMIT UNIVERSITY

Admin

- Assignment 2
  - Presentations next week, check the timetable
  - More about this at the end of the lecture
- Exam
  - **June 21**
  - More about this at the end of the lecture

Topics

- Operator Overloading (continued)
- Lambda Functions
- Code Re-use
- Extraneous Copying

Operator Overloading

RMIT UNIVERSITY

Operator Overloading

- Permits classes to make use of the 38 C++ operators, including:
  - Comparison operators: ==, !=, <, <=, >, >=
  - Arithmetic operators: +, -, *, /, %, ^, +=, -=, etc.
  - Increment/Decrement: ++, --
  - Assignment operator: =
  - I/O operators: <<, >>
  - Access: [], *, ->, &
- Generally operators can be divided into:
  - Operators that do not modify the class
  - Operators that modify the class, and return it
  - Operators that return a new class
- https://en.cppreference.com/w/cpp/language/operators

Operator Overloading

- Operators can be overloaded through:
  - Methods (member functions) on a class
  - Functions external to the class
- For this course we will stick to member functions, though the concept for function versions is very similar

Operator Overloading

- In an expression:

```
lhs <operator> rhs
```

- The overloaded operator method is called on the object on the left-hand side
  - The object on the right-hand side is passed as a parameter

Comparison Operators

- All comparison operators take the form:

```
bool operator???(const Class& rhs) const;
```

  - They return a bool
  - Take the RHS as a constant reference
  - Are a const method
- You have to implement any comparison operator you wish to use:
  - But, only really need to implement two:

```
bool operator==(const Class& rhs) const;
bool operator<(const Class& rhs) const;
```

  - The other comparison operators are implemented from these

Access Operator

- The square bracket access operator[]

```
type& operator[](const int index);
```

  - Note how it returns a reference!
  - This is a reference to the index being accessed
  - Allows the value at the index to be updated

Arithmetic Operators (self modifying)

- Some arithmetic operators directly modify the object itself:

```
Object& operator+=(const Object& other);
Object& operator+=(int value);
```

  - Still return a reference to the current object
    - This is for operator chaining reasons
    - ALL c++ operators return a value - none are void
  - The parameter can be of any desirable type to add to the object

Arithmetic Operators (new)

- Some arithmetic operators generate a new object:

```
Object operator+(const Object& other) const;
```

  - Generate a new object
    - The new object is the result of the arithmetic operation
  - Does NOT modify the object itself
    - This is why it is const

Assignment Operator

- Replaces the contents of the current object with a COPY of the contents of the object passed as the parameter:

```
Object& operator=(const Object& other);
```

  - Differs from a copy constructor as this modifies an existing object
  - Returns a reference to the current object (after modification)

Increment Operators

- Pre-increment:

```
Object& operator++();
```

  - Modifies the existing object
  - Returns a reference to the existing object

- Post-increment

```
Object operator++(int);
```

  - COPIES the object first
  - Then increments
  - Then returns the copy (not it is not a reference)
  - Requires a different parameter (which is ignored) to permit the overloading

Stream Operators

- These MUST be defined as non-member functions. That is, they cannot be methods of the class, but defined outside of the class.

```
std::ostream& operator<<(std::ostream& os, other& vec);
```

  - Outputs the operator to the provided output stream
  - Returns a reference to the stream that was written to

Lambda Functions

RMIT UNIVERSITY

Lambda Functions

- Lambda Functions are function that sit within the scope of another function
- They allow for:
  - Local code re-use to avoid duplication of the same logic
  - Prevent are large scope or namespace (such as a class or C++ file) to be “polluted” by additional functions/methods which should really be contained to a small local scope
- Issues include:
  - Should be kept small and simple, otherwise they causes code-readability issues
  - May result in code duplication (of the same lambda function) across a large scope/file since non of those methods export the duplicated logic

Lambda Functions

- Lambda Functions work identically to normal function/methods, with one exception
  - The function can “capture” variables from the scope of the external function
  - Variables can be captured:
    - By explicit name (as a copy or reference)
    - By default (capture all external variables that are used)
      - [=] by copy
      - [&] by reference

```
auto name = [capture]( params ) {
    ...
};
```

Code Re-use

RMIT UNIVERSITY

Class: Code Re-use

- Where possible, a derived class should re-use as much code as possible from it's base class(es), rather than overriding and re-implementing methods
- Things to consider include:
  - If a method requires no changes, it should not be overridden
  - An overriden method should call the base class version for common code
  - If multiple derived classes use similar logic, that should be placed as a protected method in the base class

Polymorphism: Code Re-use

- The most appropriate class should be used in the code
- Things to consider include:
  - A good principle is to use the most “general” base class for any block of code
    - This is determined by what methods need to be called
    - What other code may be passed the object
  - Typecasting should be avoided

Generics: Code Re-use

- Place logic that is identical, except for the types involved, into a single place
- Allow the compiler to generate the necessary code for each type

Operator Overloading: Code Re-use

- Allow re-use of standard operators and concepts when writing code
  - Such as assignment, comparison, I/O, etc.
- Reduces the need to manually specify custom functions/methods
  - See Java .equals(), etc.
- Can make code more easy to “intuitively” interpret
  - Such as `operator+()` on `std::string`

Function: Code Re-use

- Have seen in many courses the purpose of modularity
- Splitting code into functions allows those functions to be re-used, rather than re-written

Lambda Function: Code Re-use

- Allow the principle of function re-use, but keep that re-use to a localised scope

What does the assignment operator (=) do?

RMIT UNIVERSITY

Assignment 2 Presentations

RMIT UNIVERSITY

Assignment 2 Presentations

- 10 minutes for your group to present
- 5 minutes for questions
- Be on time
- Be prepared to start when your time
  - Don’t waste time getting your machine open, code, terminal, etc.

Assignment 2 Presentations

- What you do is up to you
- You are demonstrating your work
  - Your software
  - Your enhancements
  - Your testing
  - Your code quality & design
- For each marking criteria, convince your marker(s) of the grade you should get
- You MUST demonstrate what your group submitted
  - If you demonstrate something else, this is academic misconduct

Exam

RMIT UNIVERSITY

Exam

- June 21
- 2 hours (+15 minutes reading)
- 1 A4 page of self-prepared notes (printed, hand-written, and/or annotated), double-sided. All Notes must be collected at the end of the exam.
- Format
  - See sample layout
  - 4 Sections
  - Variety of style of questions, including short answer, coding, explanation
  - Format of these is similar to:
    - Self-Revision Questions
    - Tutorial Questions
    - Lab Questions

Exam

- Everything in the course is assessable
  - This includes today's topics
  - The only exception is the use of Hoare Logic, week 7
- How to revise/practice:
  - Weekly self-revision questions
  - Tutorial questions
  - Lab exercises
- Exam consultation: date TBA, closer to the exam

- Will a sample exam be released?
  - No

RMIT
UNIVERSITY