CM2204: Advanced Programming
Laboratory Worksheet (Week 10)

Prof. D. Marshall

Aims and Objectives

After working through this worksheet you should be familiar with:

- Understand the purpose of Templates
- Write simple classes & functions that use Templates
- Be able to overload functions in subclasses
- Understand C++ exceptions and catch exceptions in C++ code.

None of the work here is part of the assessed coursework for this module.

- Follow the web links for files highlighted and underscored to get code listings
- All lecture and lab class code is a available on the CM2204 Web page
- Solutions to the exercises will be released on the CM2204 Web page in **Week 10**.

Templates, Operator Overload & Exceptions

1. **Templates**: Define template to implement a Stack:

    - you may use the C++ `vector<T>` class to implement the element storage
    - Create methods to `push()` and `pop()` data to and from the stack.
    - Using the template,

        (a) create an instance of a Stack that can store `int`*egers*

        (b) create another instance of a Stack that can store `string`*s*

    - Demonstrate you code working with some simple examples of creating, `push`ing and `pop`ping integers and strings off the respective stacks.

2. **Operator Overloading**: Add to the `complex.cpp` example to include to following:

    - An overloaded assignment `=` operator
    - An overloaded subtraction `-` operator
    - An overloaded multiplication `*` operator
    - An overloaded division `/` operator

    For a complex numbers $z_1 = a_1 + b_1 i$:

    - assignment for a new number $z_2 = a_2 + b_2 i$ is defined as $a_2 = a_1$ and $b_2 = b_2$.

    For two complex numbers $z_1 = a_1 + b_1 i$ and $z_2 = a_2 + b_2 i$:

    - Subtraction is defined as

    $$z_1 - z_2 = (a_1 - a_2) + (b_1 - b_2)i$$

    - Multiplication is defined as

    $$z_1 * z_2 = (a1 * a_2 - b_1 * b_2) + (a1 * b_2 + a_2 * b_1)i$$

    - Division is defined as

    $$z_1 * z_2 = \frac{a1 * a_2 + b_1 * b_2}{a_2 * a_2 + b_2 * b_2} + \frac{a_2 * b_1 - a1 * b_2}{a_2 * a_2 + b_2 * b_2} i$$

    Test you program by writing a main that can manipulate and print out a few complex numbers.

3. **Operator Overloading** Write a class `Broken_maths` that perfumes integer arithmetic except that the `operator+` is overloaded to perform subtraction and the `operator-` is overloaded to perform addition! *Not recommend but possible in* C++

4. **Exceptions**: Catch a division by zero exception (`overflow_error`) and report appropriately.

5. (Question 4 of Chapter 7 of Thinking in C++, *Vol. 2*) **Exceptions**: Create a class with its own operator `new`.

   - This operator should allocate 10 objects, and on the 11th *"run out of memory"* and `throw` an **exception**.
   - Also add a `static` member function that *reclaims* this memory.
   - Now create a `main()` with a `try` block and a `catch` clause that calls the memory restoration routine.
     - Put these inside a `while` loop, to demonstrate recovering from an `exception` and continuing execution.

Further Practice

1. (Question 14 of Chapter 12 of Thinking in C++, *Vol. 1*)
   Operator Overloading: Write a class called `Bird` that contains a `string` member and a `static int`.

   - In the default constructor, use the `int` to automatically generate an identifier that you build in the string, along with the name of the class (*i.e. Bird no.1, Bird no. 2, etc.*).
   - Add an `operator<<` for `ostreams` to print out the `Bird` objects.
   - Write an assignment `operator=` and a `copy` constructor.
   - In `main()`, verify that everything works correctly.

2. (Question 3 of Chapter 7 of Thinking in C++, *Vol. 2*) **Exceptions**: Write a generic `main( )` that takes all exceptions and reports them as errors.

3. (Question 5 of Chapter 7 of Thinking in C++, *Vol. 2*) **Exceptions**: Create a destructor that throws an exception, and write code to prove to yourself that this is a bad idea by showing that if a new exception is thrown before the handler for the existing one is reached, `terminate()` is called.