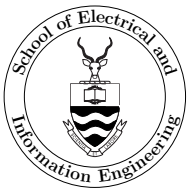


UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
School of Electrical and Information Engineering
Software Development II – Laboratory 1

An Introduction to Objects and Classes

1 Introduction

The main purpose of this laboratory is to introduce object-based programming. This is done in stages, firstly, you are required to simply make use of an existing class/type from the Standard Template Library (STL). Secondly, you need to modify, and enhance, a partially-written `Screen` class. Finally, you are in a position to create your own `Stopwatch` class from scratch.

The main laboratory outcomes are:

1. You are able to construct and use objects including those from the STL as well as programmer-defined classes.
2. You are able to modify and create new member functions for an existing class, as well as, create your own classes.
3. You consider class design with regard to the suitability of member functions.
4. You consider the differences between a class's interface and its implementation.

The first exercise of the laboratory is provided to help you revise basic C++ concepts that have been covered in Software Development I. It is advisable that you review your notes from that course before attempting the exercise.

2 C++ Revision

Exercise 2.1

Write a program which gives the user 5 chances to guess a secret random number (between 1 and 100). If they guess the number they win the game and it ends; otherwise they lose after 5 guesses. After each guess an appropriate message is to be shown: "Guess higher", "Guess lower", "You win" or "You lose".

To generate a pseudo-random number in C++ use the `rand()` function which is part of the `cstdlib` header file. `rand` generates a random number between 0 and the constant `RAND_MAX` (inclusive). How can you determine the value of `RAND_MAX`? To scale the random number to the right range make use of the modulus or "remainder from division" (`%`) operator.

The `rand()` function generates *pseudo*-random numbers because the sequence of numbers, although random, repeats itself each time the program is executed. This is because the *seed* for the random number generator is identical each time. In order to generate a completely random sequence of numbers each time the program is run it is necessary to seed the random number generator with a unique seed for each run. One way of doing this is to seed the generator with a number based on the current time using the following code: `srand(time(0));` The `time` function returns the current calendar time in seconds and is part of the `ctime` library. Note that `srand` need only be called once in a program to have the desired randomising effect.

3 The `complex` Type

The STL includes a *template* class which represents complex numbers. More details on this class can be found at `https://msdn.microsoft.com/en-us/library/0352zzhd.aspx`. Examine Listing 1 to see how objects of this class are constructed and used.

```
// complex.cpp
// Multiplying complex numbers

#include <iostream>
#include <complex> // required for the complex class

using namespace std;

int main()
{
    complex<float> num1{ 2.0, 2.0 };  // using C++11 uniform initialisation syntax
    complex<float> num2{ 4.0, -2.0 }; // old syntax: complex<float> num2(4.0,-2.0);

    auto answer = num1 * num2; // using C++11 auto keyword,
                               // answer is of type: complex<float>

    cout << "The answer is: " << answer << endl;
    cout << "Or in a more familiar form: " << answer.real() << " + " <<
        answer.imag() << "j" << endl;

    // answer++;

    return 0;
}
```

Listing 1: Complex Numbers

A template class usually requires the programmer to supply a type for the class. In the case of the `complex` class, the type supplied is the type that is used for storing the real and imaginary parts of the complex number. In Listing 1 `complex` is instantiated with the `float` type to creating floating point complex numbers. Alternatively, we could have used another numeric type such as `int` or `double`.

Why do you think the commented line of code does not work?

Exercise 3.1

We can improve the readability of the above code slightly by using an *alias* for the rather unwieldy: `complex<float>`. Read up on the `using` keyword at `https://msdn.microsoft.com/en-us/library/dn467695.aspx` and rewrite Listing 1 using a more readable name for `complex<float>`.

Exercise 3.2

The solution of a quadratic equation of the form

$$ax^2 + bx + c = 0$$

is given by

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

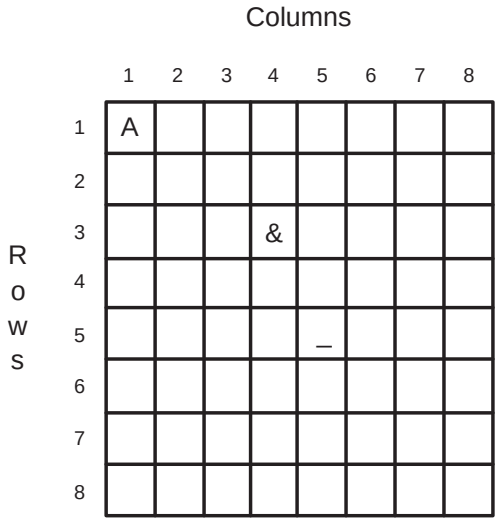
Write a program that requests the integers a, b and c from the user, calculates the roots and displays the result. The program must then ask the user if they wish to do another calculation and repeatedly calculate roots for different sets of constants until the user presses "q" to quit. You are required to determine the roots even if they happen to be imaginary.

Hint: a good way of solving this problem is to leverage the functionality of the `complex` type by making extensive use of it, rather than reverting to using primitive types (`float` etc).

You may assume that the user will not make any input errors such as supplying a character when a number is required. This is not really a reasonable assumption but we will only cover error checking later in the course.

4 Modelling a `Screen`

Consider the concept of a screen which is composed of text characters.



The screen consists of a grid. Each position in the grid is indexed by specifying the appropriate row and column. For example, "A" is at position (1,1), while the ampersand is at position (3,4). A cursor is used to identify the position on the screen that will next be read from or written to. So in the diagram above, the next character to be written to the screen will be placed at position (5,5). Note that the cursor is never actually displayed.

This concept or abstraction is captured by the `Screen` class. The screen module (`screen.h` and `screen.cpp`), as well as, a `main` function which exercises a `Screen` object can be downloaded from the course home page.

Exercise 4.1

Create a project with the above three files and run the main program which demonstrates the capabilities of a `Screen` object. Relate the output that is generated, to the source code of the `main` function and `screen.h`.

Resisting the temptation to examine `screen.cpp`, use the member functions declared in `screen.h` to generate a 6×6 screen that contains your first initial. For example, if your initial is "S" then you could generate the following:

```
******
*
*
******
     *
******
```

Exercise 4.2

Now open `screen.cpp` and try to understand the source code of the various member functions. `Screen` is implemented using the `string` class so it will be helpful to review the reference sheet for the `string` class (on the course web site) or the online documentation at `https://msdn.microsoft.com/en-us/library/syxtdd4f.aspx`.

Identify *three* different situations in which the `const` keyword is being used, and explain the meaning of `const` in each of these situations.

Also investigate the meaning of string class's `at` member function (used in `Screen::reSize`).

Exercise 4.3

We wish to provide an additional `move` function which accepts a "direction". The direction can take on one of the following values:

- HOME
- FORWARD
- BACK
- UP
- DOWN
- END

Overload the existing `move` function with a function having the following declaration:

```
void move(Direction dir);
```

where `Direction` is a scoped or strongly-typed enumeration. Read up on strongly-typed enumerations and prefer to use these over pre-C++11 enumerations.

Note that this function can be implemented entirely in terms of existing functionality, which is what you should do.

> Follow the DRY principle — Don't Repeat Yourself

Is this member function a necessity for clients of `Screen`?

Exercise 4.4

The `Screen` functions `forward` and `back` wrap around. For example, if the cursor is positioned at the bottom right-hand position of the screen, a call to `forward` will cause the cursor to advance to the top left-hand position of the screen.

However, the functions `up` and `down` do not wrap around and report errors if used when the cursor is positioned on the top and bottom rows of the screen, respectively. It is important to offer the client of a `Screen` object a consistent interface. "Wrap-around" functionality should be provided for all relative movement functions. Implement this functionality for both `up` and `down`.

Exercise 4.5

We would like the ability to draw empty squares on our screen. Create a member function that accepts the co-ordinates of the top-left corner of the square as well as the length of the sides and draws the square on the screen. Ensure that your function provides proper error checking. This function is relatively complex and it may be a good idea to implement it by making use of other `private` helper functions.

Do you need access to the internal representation of the `Screen` class in order to implement this function or can you simply use the existing interface? Does a function like this really form part of the responsibilities of a `Screen` object? Give reasons for your answer.

Exercise 4.6

The internal representation of the `Screen` class as a `string` is perhaps not as intuitive as it could be, resulting in member functions that are not easily understandable. Can you think of a better (more intuitive) internal representation that could be used without the existing `public` interface having to change?

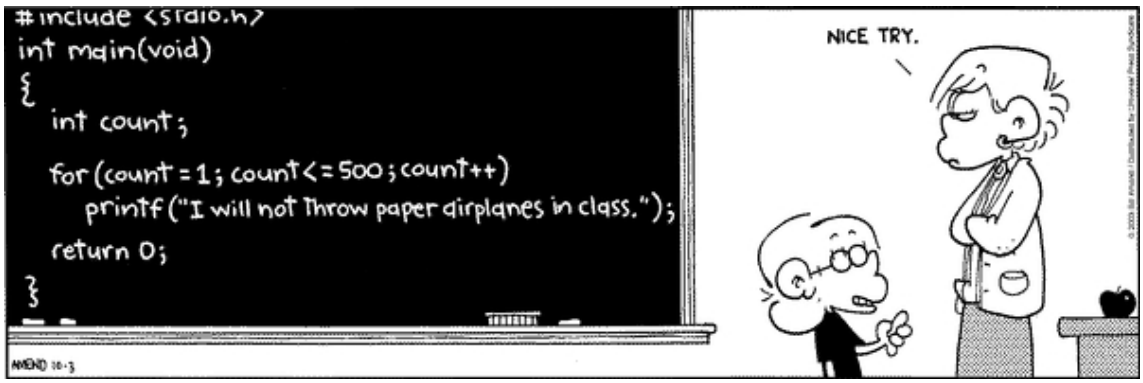
Why is it important to avoid changing the class's interface, and why are we free to change the implementation?

5 Modelling a Stopwatch

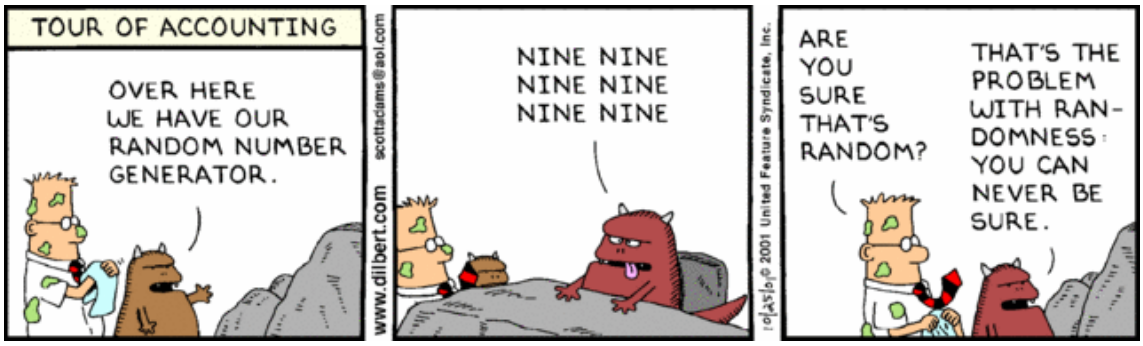
Exercise 5.1

Model a simple stopwatch which times, in seconds, how long a section of code takes to execute. Think carefully about how a stopwatch behaves and write down its responsibilities *before* deciding on its interface and implementation.

Implement the stopwatch in the files `StopWatch.h` and `StopWatch.cpp`, and test it. The function `getProcessTime` is provided which allows you to obtain the amount of time (in seconds) that has passed since your program started executing. You should make use of this function within your class.



Source: `http://www.foxtrot.com/`



Source: `http://dilbert.com/strips/comic/2001-10-25/`