Portfolio 1 - C++

Sam Bowyer

2023-05-29

Basic C++

Context

C++ is a very popular programming language, and is used in many different contexts. As a compiled language, it is often used for low-level programming, such as writing drivers for hardware devices, or for writing operating systems, however, there are also many high-level libraries written in C++ which are used in many different contexts, such as the Qt framework for GUI development, or the OpenCV library for computer vision.

One of the main advantages of C++ is that it is a very fast language, which we can use to our advantage even from within other languages, for example, in `R` using `Rcpp` or in `Python` using `Cython`.

In this portfolio we will look at the basics of using `C++` to write some simple programs.

Hello World

We can define this very simple program in a file called `hello.cpp`:

```
#include <iostream>

int main()
{
    std::cout << "Hello World!" << std::endl;

    return 0;
}
```

The different parts of this program are:

- `#include <iostream>`: include the `iostream` library so that we can print to the console
- `int main()`: the main function of the program—every program you want to run directly must have a `main()` function
- `std::cout << "Hello from C++" << std::endl;`: print "Hello World! to the console
- `return 0;`: return 0 to the operating system to indicate that the program ran successfully

This can be compiled in the command line with the command:

```
g++ hello.cpp -o hello
```

and run with:

```
./hello
```

```
## Hello from C++
```

(Note that we don't need to specify the `.cpp` extension when running the program since the compilation creates an executable file called `hello` (with no file extension).)

NOTE: Since C++ is a compiled language, you need to compile it before you can run it, as opposed to interpreted languages like Python or R which you can run directly as they are converted to machine code on the fly.

A more complex example

The second example we'll look at simply produces times tables, we will call this using the file `times.cpp`:

```
#include <iostream>

#include "timestable.h"

int main()
{
    std::cout << "Five times table" << std::endl;
    timestable(5,10);

    std::cout << "Twelve times table" << std::endl;
    timestable(12,10);

    return 0;
}
```

Here we're going to use a function defined in another file, `timestable.cpp`, which we tell `main.cpp` about with the `#include "timestable.h"` line. The header file `timestable.h` looks like:

```
#ifndef _TIMESTABLE_H
#define _TIMESTABLE_H

/* Function to print the times table for a given number
 * @param n the number to print the times table for
 * @param m the number of rows to print
 */
void timestable(int n, int m);

#endif
```

The two lines at the top are called `#define` statements, which are used to define constants in C++, this stops the compiler from trying to compile the same file twice. The `#ifndef` statement checks if the constant `_TIMESTABLE_H` has been defined, and if it hasn't, it defines it and then compiles the rest of the file. If it has been defined, it skips the rest of the file (due to the `#endif` at the bottom of the file).

The line `void timestable(int n, int m);` is a function declaration, which tells the compiler that there is a function called `timestable` which takes two integers as arguments and returns nothing (hence the `void` return type).

Also note that we have use a multi-line comment which in `C++` are defined with `/*` and `*/`, whereas single line comments are defined with `//`.

The actual implementation of the function is in `timestable.cpp`:

```
#include <iostream>

#include "timestable.h"

void timestable(int num, int maxMultiple){
    for (int i=1; i<=maxMultiple; i++){
        std::cout << i*num <<std::endl;
```

```
    }
}
```

Again we include the header file `timestable.h` so that the compiler knows what the function declaration is. The function is then defined with a `for` loop, the syntax of which is `for (initialisation; condition; increment)`, where the `initialisation` is executed once at the start of the loop, the `condition` is checked before each iteration of the loop, and the `increment` is executed after each iteration of the loop. In this case, the `initialisation` is `int i=1`, the `condition` is `i<=maxMultiple`, and the `increment` is `i++`.

We can also use a `while` loop, which is defined as `while (condition)`, where the `condition` is checked before each iteration of the loop.

We compile the program with

```
g++ times.cpp timestable.cpp -o times
```

and run it with `./times` (note that we include the `timestable.cpp` file in the compilation command so that `g++` knows to use it too).

```
./times
```

```
## Five times table
## 5
## 10
## 15
## 20
## 25
## 30
## 35
## 40
## 45
## 50
## Twelve times table
## 12
## 24
## 36
## 48
## 60
## 72
## 84
## 96
## 108
## 120
```

Variables and OOP

C++ is often used in conjunction with object-oriented programming (OOP), which is a programming paradigm which uses objects to represent real-world entities, and which uses classes to define the properties and methods of these objects. For our final example we will then implement a simple class to represent patient data in a hospital. This will also allow us to look at some of the different variable types in C++.

The class will be defined in a file called `patient.h`:

```
#ifndef _PERSONDATA_H
#define _PERSONDATA_H

/* The data about a person in the database */
class PersonData
```

```
{
public:
    PersonData();

    void setHeight(float height);
    void setWeight(float weight);

    float height();
    float weight();
    float bmi();

private:
    /* The person's height */
    float _height;

    /* The person's weight */
    float _weight;
};

#endif
```

We can see that a class is defined using the `class` keyword, and that the class is called `PersonData`. This will contain two private variables, `_height` and `_weight`, which are of type `float`, and two public functions, `setHeight` and `setWeight`, which are used to set the values of the private variables, and two public functions, `height` and `weight`, which are used to get the values of the private variables. (It is common OOP practice to prefix private variables with an underscore, to distinguish them from public variables and also to implement getters and setters (functions to read and write variable values) for private variables, to allow for more control over how they are accessed.)

Also note that the `PersonData` class has a constructor, which is defined as `PersonData();`, and which is called when a new instance of the class is created.

The implementation of the class is in `patient.cpp`:

```
#include "persondata.h"

PersonData::PersonData()
{
    this->_height = 0;
    this->_weight = 0;
}

void PersonData::setHeight(float height)
{
    this->_height = height;
}

void PersonData::setWeight(float weight)
{
    this->_weight = weight;
}

float PersonData::height()
{
    return this->_height;
```

```
}

float PersonData::weight()
{
    return this->_weight;
}

float PersonData::bmi()
{
    return this->_weight / (this->_height * this->_height);
}
```

We can see that the constructor is defined as `PersonData::PersonData()`, where `PersonData` is the name of the class, and `PersonData()` is the name of the constructor. The other functions are defined in a similar way, with the class name followed by `::` followed by the function name.

We can then use this class in a program called `main.cpp`:

```
#include <iostream>
#include <map>
#include <string>
#include <vector>

#include "persondata.h"

int main()
{
    //declare the map that uses a person's name as a key to look
    //up their personal data stored in the PersonData object
    std::map<std::string,PersonData> database;

    //let's first put the data in three vectors
    std::vector<std::string> names = { "James", "Jane", "Janet", "John" };
    std::vector<float> heights = { 1.7, 1.8, 1.5, 1.4 };
    std::vector<float> weights = { 75.4, 76.5, 56.8, 52.0 };

    //now put all of the data into the database
    for (int i=0; i<names.size(); ++i)
    {
        PersonData data;
        data.setHeight( heights[i] );
        data.setWeight( weights[i] );

        database[names[i]] = data;
    }

    //now print out the entire database
    for ( auto item : database )
    {
        //print out the name
        std::cout << item.first << " : ";

        auto data = item.second;

        std::cout << "height=" << data.height()
```

```
                  << " weight=" << data.weight()
                  << " bmi=" << data.bmi() << std::endl;
    }

    return 0;
}
```

This program creates a map called `database` which uses a `std::string` as a key to look up a `PersonData` object. These `PersonData` objects are then created and populated with data that is defined in three vectors: `names`, `heights`, and `weights` (note that these vectors are defined as `std::vector` objects, which is a type of container that can be used to store multiple values of the same type).

The program then adds the data from the vectors to the map in the first `for` loop, before printing out the data in the second `for` loop.

Interestingly, the `for` loop in the second `for` loop is a `range-based for loop`, which is defined as `for ( auto item : database )`, where `auto` is a keyword that tells the compiler to automatically determine the type of the variable `item`, and `database` is the container that we want to iterate over. This `auto` keyword is useful for when we want to iterate over a container, but we don't care about the type of the container, or when we want to iterate over a container of a type that we don't know—we also use it in when we access the elements of our map in the line `auto data = item.second;`.

Similarly to the last example, we compile this program using:

```
g++ -std=c++11 main.cpp persondata.cpp -o main
```

(The `-std=c++11` flag is used to tell the compiler that we want to use C++11 features, which is required for the `auto` keyword.)

We can then run the program using:

```
./main
```

```
## James : height=1.7 weight=75.4 bmi=26.09
## Jane : height=1.8 weight=76.5 bmi=23.6111
## Janet : height=1.5 weight=56.8 bmi=25.2444
## John : height=1.4 weight=52 bmi=26.5306
```

Summary

Here we have looked at some of the basic syntax of C++, including how to compile and run a program, how to use variables, how to use functions, and how to use classes. We have also looked at some of the different variable types in C++, including `int`, `float`, `double`, `bool`, `char`, `std::string`, `std::vector`, and `std::map`.

In the proceeding portfolios we will use C++ in conjunction with `R` using `Rcpp` to speed up the execution of our code, as well as dicuss how we can speed up our `C++` code via parallelisation with OpenMP and Intel-TBB.