portfolio_1

Henry Bourne

2023-01-30

Intro to C++

In this document we will go over the basics of C++, the idea of this document is for it to act as a quick reference sheet to refresh on C++ syntax should I forget. We will use bash to print out and run C++ files all of which (along with this R markdown file can be found on my github: "github.com/h-aze/compass_yr1/SC2"). Some explanation I will give before the code chunks and some I will give in the comments of the C++ files.

The basics

We will start by covering the basics which is all contained in the "intro.cpp" file, lets print it out and run it:

```
cat intro_to_cpp/intro.cpp
```

```
## #include <iostream>
## #include <string>
## // We will explain what the above mean shortly
##
## // This chunk of code describes how to define a function aswell as some useful syntax,
## // read this once you have gotten to where a_function is called in main!
## // Again everything in c++ is typed including functions
## // if we didn't want to return anything we could set the type to "void"
## double a_function(int arg)
## {
##     // We can condition using the following syntax and print out something useful about arg
##     if (arg <0){
##         std::cout << arg << " is negative." << std::endl;
##     }
##     else if (arg < 20){
##         std::cout << arg << " is between 0 and 20." << std::endl;
##     }
##     else{
##         std::cout << arg << " is bigger than 20" << std::endl;
##     }
##
##     // We can carry out a for loop as follows which prints out the numbers from 10 to 1
##     // We could also write i=i-1 as i-- or i-=1
##     for (int i=10; i>0; i=i-1){
##         std::cout << i << std::endl;
##     }
##
##     double out = arg * 0.01;
##     return out;
## }
```

```
##
## // We now define a function main and put all our code inside,
## // everything inside main will then be run when we run this code
## int main()
## {
##     // In C++ everything is typed and so when we define anything we must make sure we type it,
##     // eg. we can define a variable that is the integer 1 as follows:
##     int a = 1 ;
##
##     // Notice that we put a ";" after as this is how c++ defines line breaks,
##     // if we wanted to define a string we could do the following:
##     std::string our_string = "Hello!";
##     std::cout << our_string << std::endl;
##
##     // Here we define a variable our_string that is of type string (std::string)
##     // and then print the output.
##     // We use "#include <string>" to include the string header file
##     // which allows us to use the string type.
##     // We also type "#include <iostream>" which gives us the functionality
##     // of getting inputs and giving outputs,
##     // for example if we want to print to the console we need to include this header file.
##     // We can then use std::cout and std::endl as above to print to the console.
##     // Other things we can do with this header file is throw exceptions and return errors,
##     // we will write how to do this below but comment them out
##     // so we don't get any errors when running this code:
##     // std::cerr << "Some error" << std::endl;
##     // throw std::runtime_error("Unknown exception");
##
##     // We now call the function "a_function" which we define above.
##     // Go read that portion of code now!
##     a_function(10) ;
##
##     return 0;
## }
```

```
g++ intro_to_cpp/intro.cpp -o intro_to_cpp/intro
./intro_to_cpp/intro
```

```
## Hello!
## 10 is between 0 and 20.
## 10
## 9
## 8
## 7
## 6
## 5
## 4
## 3
## 2
## 1
```

Types

Let's now discuss types in more detail:

```
cat intro_to_cpp/types.cpp
```

```
## // We first include header files we will need in this script
## #include <iostream>
## #include <string>
##
## int main(){
##     // Let's first define a variable of type double
##     double var = 10.5;
##     std::cout << "Var as a double: " << var << std::endl;
##
##     // We can change the type of the variable as follows:
##     int var1 = var;
##     std::cout << "Var now converted to int: " << var1 << std::endl;
##
##     // If you check the output of running this file
##     // you can see we have lost information
##     // So must be careful when converting types
##
##     // We also have auto:
##     // if it is obvious what type a var or fun should have,
##     // you can use "auto" eg,
##     auto var2 = var1 ;
##     // ^ Here it is "obvious" that var2 should be of type int
##     std::cout << "Var assigned using auto: " << var2 << std::endl;
## }
```

```
g++ intro_to_cpp/types.cpp -o intro_to_cpp/types
./intro_to_cpp/types
```

```
## Var as a double: 10.5
## Var now converted to int: 10
## Var assigned using auto: 10
```

Scope

Scope is controlled by "{}", anything defined in the brackets is only available in the brackets. It is also defined by files, anything defined in a file is only available in that file, unless we import of course (we will discuss this later). Note that if we have multiple definitions of a variable in different levels of nests we use the definition of the inner most nest we currently are in.

Vectors and Dictionaries

```
cat intro_to_cpp/vector_dict.cpp
```

```
## // Don't read this for now ...
## // --------------------------------------
## #include <iostream>
## #include <vector> // We need this to use vectors
## #include <map> // We need this to use dictionaries (maps in C++)
##
## // This function creates a vector of vectors (a matrix)
## std::vector<std::vector<int>> get_nested_vector(){
##     std::vector< std::vector<int> > M;
```

```
##
##     // We create a vector of vectors
##     // ie. a matrix
##     for (int i=1; i<=3; ++i)
##     {
##         // We first create a row
##         std::vector<int> row;
##
##         for (int j=1; j<=3; ++j)
##         {
##             row.push_back( i * j );
##         }
##
##         // Now we save the row to the matrix
##         M.push_back(row);
##     }
##     return M;
## }
##
## // ----------------------------------------
## // START READING HERE!
## int main(){
##     // First we will talk vectors,
##     // a vector is a container that can store multiple values of only one type
##
##     // We can create a vector that can hold integers as so:
##     std::vector<int> v;
##
##     // We can add values onto the end as so:
##     v.push_back( 4 );
##     v.push_back( 2.5 );
##     // Note for the second command 2.5 will be converted to an integer
##     // Let's now print out our vector,
##     // note we use a range-based for loop here (needs C++11 or later)
##     std::cout << "Our vector: { " ;
##     for (auto i: v)
##         std::cout << i << ' ';
##     std::cout << "}" << std::endl;
##
##     // We can get the size of the vector
##     int length = v.size();
##     std::cout << "The vector has length: " << length << std::endl;
##
##     // Is also possible to nest vectors inside vectors
##     auto M = get_nested_vector();
##     // go to the get_nested_vector to see how we create and fill a matrix!
##     std::cout << "A nested vector: " << std::endl;
##     // this prints the matrix out
##     for (int i=0; i<3; ++i)
##     {
##         for (int j=0; j<3; ++j)
##         {
##             std::cout << M[i][j] << " ";
##         }
```

```
##
##         std::cout << std::endl;
##     }
##
##     // ----------
##     // Now lets talk dictionaries,
##     // dictionaries in C++ are called maps,
##     // they are containers that store key value pairs,
##     // note that keys must be of the same type and values must be of the same type
##
##     // We create a map that stores strings and where the keys are strings
##     std::map<std::string, std::string> dict;
##
##     // we can add some items to the map
##     dict["key_1"] = "value_1";
##     dict["key_2"] = "value_2";
##
##     // Now we can loop through all of the key-value pairs
##     // in the map and print them out
##     for ( auto item : dict )
##     {
##         //item.first is the key
##         std::cout << "Getting the key using .first :"
##         << item.first << " , ";
##
##         //item.second is the value
##         std::cout << "Getting the key using .second :"
##         << item.second << std::endl;
##     }
##
##     // Finally we can lookup values by key
##     std::cout << "What's the value associated to key_1?: " << dict["key_1"]
##               << std::endl;
##
##
##     return 0;
## }
```

```
g++ intro_to_cpp/vector_dict.cpp -o intro_to_cpp/vector_dict
./intro_to_cpp/vector_dict
```

```
## Our vector: { 4 2 }
## The vector has length: 2
## A nested vector:
## 1 2 3
## 2 4 6
## 3 6 9
## Getting the key using .first :key_1 , Getting the key using .second :value_1
## Getting the key using .first :key_2 , Getting the key using .second :value_2
## What's the value associated to key_1?: value_1
```

Multi-File Programmes

In C++ we use something called a header file (uses the ".h" extension). Header files are where you store all your function declarations and in the .cpp files you should store all your function definitions and main code.

What this achieves is a separation between interface (the header file) and implementation (the .cpp file). To use the function declarations in your cpp file you must use "#include". If we then want to use a function that was declared in a header file but defined elsewhere all we must do is "#include" the header file, this is how we use functions from standard libraries. There are two ways of "#include"-ing:

```
# include <standard_library_name>

# include 'user_defined_library'
```

Another thing to mention that we haven't yet is that you can overload functions, ie. you can create multiple instances of a function that take different types and at run time the definition of the function that has the correct types and number of arguments is used. Before moving on we will mention header guards, we must include these to prevent the situation where we copy are code in twice (if we have two includes of the same thing in our code for example), at the top of the file one should write:

```
# ifndef _FILENAME_H

# define _FILENAME_H
```

and at the bottom:

```
# endif
```

Finally we should have a file named "main.cpp" where we keep our main function.

Objects, Classes, Concepts, Default Arguments and Operators

Finally we will talk briefly about objects, classes, concepts, default arguments and operators. We can declare a class in C (in the header file!) using the following syntax:

```
#'class ClassName
#'{
#'public:
#'    ClassName(type arg); //This is the constructor (or initialization) method
#'
#'    type class_method(type arg);
#'
#'private:
#'    type an_attribute;
#'};
```

We can then define a method we have declared in a .cpp file using the following syntax:

```
#'ClassName::name_of_class_method(type arg)
#'{
#'  do something
#'}
```

If we want to change the value of an attribute we can do it as follows:

```
#'this->attribute_name = value;
```

We can initialize a class as follows:

```
#'ClassName obj(init_param);
```

And then use a method as follows:

```
#'obj.method_name(params);
```

Now we will talk about operators. For classes in C++ we can define operators, operators are functions that are added to classes to specify what code should be used when we operate on them with other classes. Examples of key operators are:

```
# ;operator+ : addition ;operator- : subtraction ;operator*
# : multiplication ;operator/ : division ;bool operator== :
# comparison equals to, ;bool operator!= : not equal to
# ;bool operator< : less than ;bool operator<= : less than
# or equal to ;bool operator> : greater than ;bool
# operator>= : greater than or equal to
```

We can declare an operator for a class like so:

```
#'ClassName operator+(ClassName arg_name);
```

And then define it like so:

```
#'ClassName ClassName::operator+(ClassName arg_name)
#'{
#'    do and return something
#'}
```

This concludes the intro to C++!

Let's Get Some C++ Practice!

To get coding some more C++ we will now move onto a project I have done in C++. The project was to create, train and test a neural network all written in C++.

Creating a Neural Network

Fitting a Neural Network