1 C++: Lecture 4 on 14 June 2020

1.1 Vectors

In order to be able to use a vector in your code, you must include it as a pre-processor directive.

```
#include <vector>
```

Vectors are a storage device that can hold many slots of memory, similar to that of an array. Arrays have a defined number of slots of memory, however. A vector has a variable size, making them preferable to arrays in most situations. Also like arrays, vectors can be multi-dimensional.

Vectors are typically defined in a manner different than that of most variables:

```
vector <int> myVect;
```

The general form of a vector is: *vector* $<$ *storagetype* $>$ *Name*;. Vectors, like arrays, can store many data types but only one.

```
 vector <bool> a;
 vector <int> b;
 vector <char> c;
 vector <double> d;
```

Since vectors have a varying size, we can add a single item at a time, using *.push_back*.

```
a.push_back(true);
b.push_back(123);
c.push_back('@');
d.push_back(2.5);
```

1.1.1 Simple Task

Store the integers 1-100 in a vector.

Solution:

```
for (int i = 1; i <= 100; ++i){
  myVect.push_back(i);
}
```

Storing data one at a time can be inefficient for small quantities, so a vector can be initialized:

```
vector<int> myVect2{ 10, 20, 30 };//Stores 10,20,30
```

Another method of initializing can be used that initializes the size of the vector, and all values in it.

```
int n = 4;
vector <int> myVect3(n,3);//stores n number of 3's
```

As we saw in the previous lecture, arrays can be multidimensional, and so vectors can as well. The defining statement embeds one vector within another.

```
vector<vector<int> > vect{ { 1, 2, 3 },
                           { 4, 5, 6 },
                           { 7, 8, 9 } };
```

Data is useless if we are unable to access it, and there are two major ways to do so.

Array Notation:

```
cout << myVect2[1] << endl;
//Should print 20 from before
```

or:

```
cout << myVect2.at(1) << endl;
```

Since the size can be varied, it is useful to be able to find it, which can be done with *.size*().

```
cout << myVect2.size() << endl;
//should output three, because myVect2 has three integers stored
```

With this we can introduce another type of loop.

1.2 Range Based For Loops

This loop takes a look at everything stored in a vector, one at a time and runs the code based on it. The number of times the code is sun is based on the size of the vector.

```
for (int i : myVect2) {
    cout << i << endl;
}
//is the same as
for (int i = 0; i < myVect2.size(); ++i){
    cout << myVect2[i] << endl;
}
```

We can also allow the compiler to determine the data type for us using *auto*.

```
for (auto i : myVect2){
    cout << i << endl;
}
```

1.2.1 Task

Have the user input some number of integers, and then tell them how many they have inputed so far.

Solution:

```
  vector <int> integers;
  while (1){
    cout << "Enter an integer:\n";
    int temporary;
    cin >> temporary;
    integers.push_back(temporary);
    if (integers.size() == 5){
      cout << "You have entered five integers. Here they are:";
      for (auto i : integers){
        cout << i << " ";
      }
      cout << "\n";
      break;
    }
  }
```

1.3 Functions

Functions can be used for many things, but they are mainly used to make code more elegant. They are defined outside the main function, and can only access global variables, or data that has been passed in. They can be called many times within the main, and every time it is called it runs the code within the function. For now, let's just look at how they are structured.

```
#include <iostream>
using namespace std;

//This is the function
bool opposite(bool unknown){
    return !unknown;
}

int main(){
    cout << opposite(true) << endl;
}
//The value printed is the one returned by the function
```

The general form of a function is as follows:

```
  type name(type name, type name, ...){
    code;
    ...
    return something;// must be the same as the type of function
  }
```

One interesting thing that can be done with functions is that they can be nested, which means the function is called within itself. These are called recursive functions, and be quite useful.

1.3.1 Recursive Functions

A classic example of a recursive function would be a Fibonacci calculator. The Fibonacci series is a series of numbers in which any number is the sum of the previous two, except for the initial two numbers, both of which are one.

```
#include <iostream>
using namespace std;

int Fib(int num){
    if (num == 1 || num == 2){
        return 1;
    } else {
        return Fib(num-1) + Fib(num-2);
    }
}

int main() {
    cout << "Enter the fibonaci number you want:\n";
    int num;
    cin >> num;
    cout << Fib(num) << endl;
}
```