Overview Q&A Downloads Announcements

Writing our First C++ Template in VS Code | C++ Tutorials for Beginners #64

In the last tutorial, we learnt about what a template is, why a template is used in programming and what its syntax is. Let's give ourselves a quick revision of everything about templates.
Long story short, a template does the same thing to a class, what a class does to the objects. It parametrizes the data type hence making it easy for us to use different classes without having to write the whole thing again and again, violating the DRY rule. Templates furthermore give our program a generic view, where declaring one template suffices the task.
Today, we'll learn to make a program using templates to give you a better understanding about its uses. I'll make the process effortless for you to learn, so, you stay calm and keep learning.
Now suppose we have two integer vectors and we want to calculate their Dot Product. This part should not be

troublesome since we have learnt pretty well the use of classes and constructors. We had learnt to write the code like the one mentioned below.

Understanding the code below to calculate the DotProduct of two integer vectors:

1. Here we declare a class vector, with an integer pointer arr.
2. We declared an integer variable to store the size.
3. We made the constructor for the integer vector. These things should be unchallenging for you by now as they have been already taught.
4. We then wrote a function which returns an integer value, to calculate the Dot Product and named it dotProduct which will take a vector as a parameter.
5. We traversed through the vectors multiplying their corresponding elements and adding it to the sum variable named d.
6. We finally returned it to the main.
7. And the output we received is this:

```
5
PS D:\MyData\Business\code playground\C++ course>
```

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

class vector
{
    public:
        int *arr;
        int size;
        vector(int m)
        {
            size = m;
            arr = new int[size];
        }
    int dotProduct(vector &v){
        int d=0;
        for (int i = 0; i < size; i++)
        {
            d+=this->arr[i]*v.arr[i];
        }
        return d;
    }
};

int main()
{
    vector v1(3); //vector 1
    v1.arr[0] = 4;
    v1.arr[1] = 3;
    v1.arr[2] = 1;
    vector v2(3); //vector 2
    v2.arr[0]=1;
    v2.arr[1]=0;
    v2.arr[2]=1;
    int a = v1.dotProduct(v2);
    cout<<a<<endl;
    return 0;
}
```

So, this was all about creating a class and an embedded function to calculate the dot product of two integer vectors. But this program would obviously fail to calculate the dot products for some different data types. It would demand an entirely different class. But we'll save ourselves the effort and the time by declaring a template. Let's see how:

Understanding the changes, we made in the above program to generalise it for all data types:

1. First and foremost, we defined a template with class T where T acts as a variable data type.
2. We then changed the data type of arr to T, changed its constructor to T from int, changed everything except the size of the vector, to a variable T. The function then returned T. This has now changed the class from specific to general.
3. We then very easily added a parameter, while defining the vectors, of its data type. And the compiler itself transformed the class accordingly. Here we passed a float and the code handled it very efficiently.
4. The output we received was:

```
6.82
PS D:\MyData\Business\code playground\C++ course>
```

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

```
template <class T>
class vector
{
    public:
        T *arr;
        int size;
        vector(int m)
        {
            size = m;
            arr = new T[size];
        }
    T dotProduct(vector &v){
        T d=0;
        for (int i = 0; i < size; i++)
        {
            d+=this->arr[i]*v.arr[i];
        }
        return d;
    }
};

int main()
{
    vector<float> v1(3); //vector 1 with a float data type
    v1.arr[0] = 1.4;
    v1.arr[1] = 3.3;
```

```
        v1.arr[2] = 0.1;
        vector<float> v2(3); //vector 2 with a float data type
        v2.arr[0]=0.1;
        v2.arr[1]=1.90;
        v2.arr[2]=4.1;
        float a = v1.dotProduct(v2);
        cout<<a<<endl;
        return 0;
    }
```

Imagine how tough it would have been without these templates, you'd have made different classes for different data types handling them clumsily increasing your efforts and proportionally your chances of making errors. So, this is a life saviour.

And learning it will only benefit you. So why not.

Thank you, friends for being with me throughout, hope you liked the tutorial. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. I hope you enjoy them. In the next tutorial, we'll be learning about further uses of a template and multiple parameters, see you there, till then keep coding.

Previous Next

CodeWithHarry