### C++ Templates: Must for Competitive Programming :

It has been quite a journey till here, and I feel grateful to have you all with me in the same. We have covered a lot in C++ and there is yet a great deal left. But we’ll make everything ahead a cakewalk together.

Today we have in the box, the most important topic for all you enthusiastic programmers, C++ templates. We’ll follow the below-mentioned roadmap:

1. What is a template in C++ programming?
2. Why templates?
3. Syntax

**What is a template in C++ programming?**

 A template is believed to escalate the potential of C++ several fold by giving it the ability to define data types as parameters making it useful to reduce repetitions of the same declaration of classes for different data types. Declaring classes for every other data type(which if counted is way too much) in the very first place violates the DRY( Don’t Repeat Yourself) rule of programming and on the other doesn't completely utilise the potential of C++.

It is very analogous to when we said classes are the templates for objects, here templates itself are the templates of the classes. That is, what classes are for objects, templates are for classes.

**Why templates?**

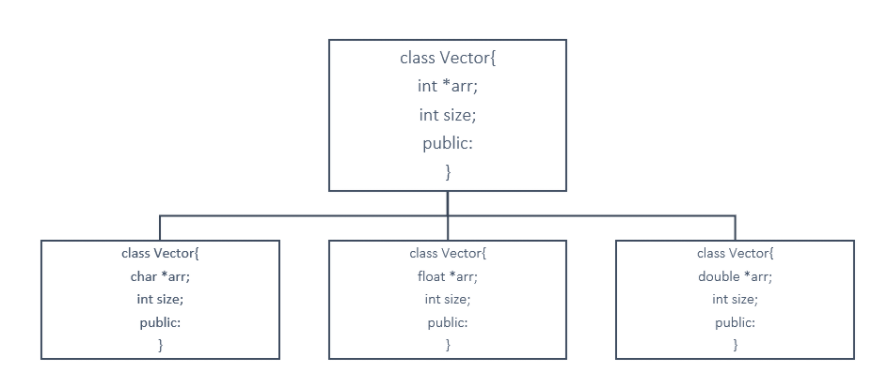
1. **DRY Rule:**

To understand the reason behind using templates, we will have to understand the effort behind declaring classes for different data types. Suppose we want to have a vector for each of the three(can be more) data types, int, float and char. Then we’ll obviously write the whole thing again and again making it awfully difficult. This is where the saviour comes, the templates. It helps parametrizing the data type and declaring it once in the source code suffice. Very similar to what we do in functions. It is because of this, also called, ‘parameterized classes’.

1. **Generic Programming:**

It is called generic, because it is sufficient to declare a template once, it becomes general and it works all along for all the data types.

Refer to the schematic below:



We had to copy the same thing again and again for different data types, but a template solves it all. Refer to the syntax section for how.

Below is the template for a vector of int data type, and it goes similarly for float char double, etc.

class vector {

int \*arr;

int size;

public:

};

**Syntax:**

Understanding the syntax below:

1. First, we declare a template of class and pass a variable T as its parameter.
2. Define the class of vector and keep the data type of \*arr as T only. Now, the array becomes of the type we supply in the template.

Now we can easily use this template to declare umpteen number of classes in our main scope. Be it int, float, or arr vector.

#include <iostream>

using namespace std;

template <class T>

class vector {

T \*arr;

int size;

public:

vector(T\* arr)[

//code

]

//and many other methods

};

int main() {

vector<int> myVec1();

vector<float> myVec2();

return 0;

}

Templates are believed to be very useful for people who pursue competitive programming. It makes their work several folds easier. It gives them an edge over others. It is a must because it saves you a lot of time while programming. And I believe you ain’t want to miss this opportunity to learn, right?

So, get to the playlist as soon as you can. Save yourselves some time and get over your competitors.

Thank you, friends, for being with me throughout, hope you liked the tutorial. And If you haven’t checked out the whole playlist yet, it’s never too late, move on to [codewithharry.com](https://www.codewithharry.com/) or my YouTube channel to access it. I hope you enjoy them. Templates are an inevitable part of this process of learning C++. You just cannot afford to miss this. In the next tutorial, we’ll be writing a program using templates for your better understanding, see you there, till then keep coding.

#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;

}

**Template With Multiple Parameters:**

template<class T1, class T2>

class myClass{

public:

T1 data1;

T2 data2;

myClass(T1 a,T2 b){

data1 = a;

data2 = b;

}

void display(){

cout<<this->data1<<" "<<this->data2;

}

};

int main()

{

myClass<int, char> obj(1, 'c');

obj.display();

}

### C++ Function Templates & Function Templates with Parameters:

template<class T1, class T2>

float funcAverage(T1 a, T2 b){

float avg= (a+b)/2.0;

return avg;

}

int main(){

float a;

a = funcAverage(5,2);

printf("The average of these numbers is %f",a);

return 0;

}

Copy

The average of these numbers is 3.500000

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

**Member Function Templates & Overloading Template Functions in C++ :**

template <class T>

class Harry

{

public:

T data;

Harry(T a)

{

data = a;

}

void display();

};

template <class T>

void Harry<T> :: display(){

cout<<data;

}

int main()

{

Harry<int> h(5.7);

cout << h.data << endl;

h.display();

return 0;

}

And the output is:

5

5

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

Copy

Now, we’ll move to the**overloading of a function template**. Overloading a function simply means assigning two or more functions with the same name, the same job, but with different parameters.  For that, we’ll declare a void function named func. And a template function with the same name. Follow the snippet below to do the same:

1. We made two void functions, one specified and one generic using a template.
2. The first one receives an integer and prints the integer with a different prefix.
3. The generic one receives the value as well as the data type and prints the value with a different prefix.
4. Now, we’ll wish to see the output of the following functions, by calling them from the main. Refer to the main program below the snippet below.

#include <iostream>

using namespace std;

void func(int a){

cout<<"I am first func() "<<a<<endl;

}

template<class T>

void func(T a){

cout<<"I am templatised func() "<<a<<endl;

}

int main()

{

func(4); //Exact match takes the highest priority

return 0;

}

And the output is,

I am first func() 4

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