**C++ Templates: Must for Competitive Programming | C++ Tutorials for Beginners #63**

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?**

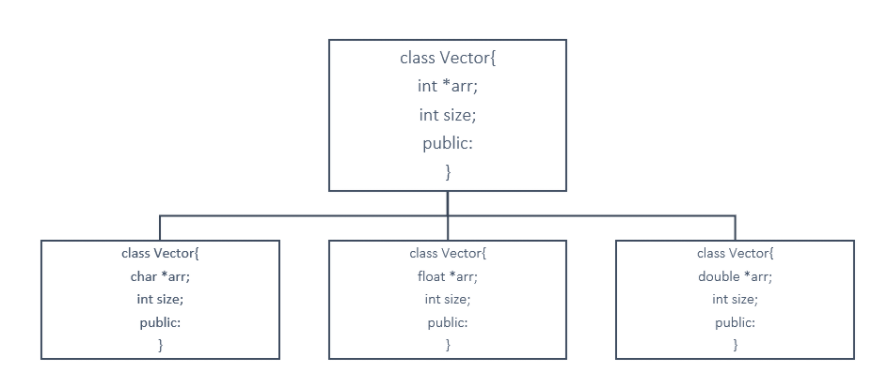
1. **DRY Rule:**

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:

};

Copy

**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;

}

Copy

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.

**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>

Copy

#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;

}

Copy

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>

Copy

#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;

}

Copy

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.

**C++ Templates: Templates with Multiple Parameters | C++ Tutorials for Beginners #65**

In the last tutorial, we had ample understanding of a template and its uses. We had created a template which would calculate the Dot Product of two vectors of any data type just by declaring a simple template parameterizing the data type we usually hardcoded in the classes. This already made our task easier but here we are, with our next tutorial focusing on how to handle multiple parameters in a template.

To give you a short overview of how templates work with multiple parameters, you can think of it as a function where you have that power to pass different parameters of the same or different data types. A simple template with two parameters would look something like this. The only effort it demands is the declaration of parameters. We’ll get through it thoroughly by making a real program, so, let’s go.

#include<iostream>

using namespace std;

/\*

template<class T1, class T2>

class nameOfClass{

//body

}

\*/

int main(){

//body of main

}

Copy

**Code Snippet 1: Syntax of a template with multiple parameter**

Suppose we have a class named myClass which has two data in it of data types int and char respectively, and the function embedded just displays the two. Fair enough, no big deal, we’ll construct our class something like this. The problem arises when we wish to have both our data types anonymous and to be put from the main itself.  You will be surprised to know that very subtle modifications in yesterday's code would do our task. Instead of declaring a single parameter T, we would declare two of them namely T1 And T2.

class myClass{

public:

int data1;

char data2;

void display(){

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

}

};

Copy

**Code Snippet 2: Constructing a class**

Refer to changes we have done below to parametrize both our data types using a single template:

1. We have declared data1 and data2 with data types T1 and T2 respectively.
2. We have applied the constructor filling the values we receive from the main into data1 and data2.
3. Finally, we have displayed both of them.

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;

}

};

Copy

**Code Snippet 2: Constructing a template with two parameters.**

Let me now show you how this template works for different parameters. I’ll pass different data types from the main and see if it's flexible enough.

Firstly, we put an integer and a char,

int main()

{

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

obj.display();

}

Copy

**Code Snippet 3: Specifying the data types to be int and char.**

And the output received was this, which is correct. Let’s feed another one.

1 c

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

Copy

**Figure 1: Output of code snippet 3.**

Now we put an integer and a float,

int main()

{

myClass<int, float> obj(1,1.8 );

obj.display();

}

Copy

**Code Snippet 4: Specifying the data types to be int and float.**

And the output received was this,

1 1.8

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

Copy

**Figure 1: Output of code snippet 4.**

So yes, this is functioning all good.

And this was all about templates with multiple parameters, just don’t miss out the commas while defining the parameters in a template. And you can have 2, 3 or more of them according to your needs. Could you believe how luxurious it has become to work with customized data types? It is now you, who’ll decide what the data type of some variable in a class should be. It is no longer pre-specified.  It has given you some unimaginable power which, if you realise, can save you a lot of energy and time.

**C++ Templates: Class Templates with Default Parameters | C++ Tutorials for Beginners #66**

So far, we have already covered the C++ templates with single parameters. In the last tutorial, we learnt about templates with multiple parameters, when it comes to handling different data types of two or more containers.

Today, we’ll be learning a very easy yet powerful attribute of templates, its ability to have default parameters. Its ability to have default specifications about the data type, when it receives no arguments from the main.

So, let's start by making a program manifesting the use of default parameters in a C++ template. **Refer to the code snippet below and follow the steps:**

1. We’ll start by constructing a class named Harry.
2. We’ll then define a template with any number of arguments, let three, T1, T2, and T3. If you remember, we had this feature of specifying default arguments for functions, similarly we’ll mention the default parameters, let, int, float and char for T1, T2 and T3 respectively.
3. This ensures that if the user doesn’t put any data type in main, default ones get considered.
4. In public, we’ll  define variables a, b and c of the variable data types T1, T2 and T3. And build their constructors.
5. The constructor accepts the values featured by the main, and assigns them to our class variables a, b and c. If the user specifies the data types along with the values, the compiler assigns them to T1 , T2 and T3, otherwise gives them the default ones, as specified while declaring the template itself.
6. We’ll then create a void function display, just to print the values the user inputs.

#include<iostream>

using namespace std;

template <class T1=int, class T2=float, class T3=char>

class Harry{

public:

T1 a;

T2 b;

T3 c;

Harry(T1 x, T2 y, T3 z) {

a = x;

b = y;

c = z;

}

void display(){

cout<<"The value of a is "<<a<<endl;

cout<<"The value of b is "<<b<<endl;

cout<<"The value of c is "<<c<<endl;

}

};

Copy

Since we are done defining the templates and class, we can very easily move to the main where we’ll see how these work. **Understanding code snippet 2:**

1. Firstly, we’ll create an object, let's name it h, of the class Harry. And we’ll pass into it three values, an int, a float and a char, suppose 4, 6.4 and c respectively. Now since we have not specified the data types of the values we have just entered, the default data types, int, float and char would be considered.
2. We’ll then display the values, which you’ll be seeing when we run the same.
3. And then we’ll create another object g, of the class Harry but this time, with the data types of our choice. Let’s specify them to be float, char and char.
4. We can then pass some values into it, suppose 1.6, o, and c and call the display function again.
5. These objects are sufficient to give us the main concept behind using a default parameter and the variety of classes we could make via this one template.

int main()

{

Harry<> h(4, 6.4, 'c');

h.display();

cout << endl;

Harry<float, char, char> g(1.6, 'o', 'c');

g.display();

return 0;

}

Copy

We’ll now refer to the output the above codes combinedly gave. As you can see below, it worked all fine. Had we not specified the default parameters; the above program would have thrown an error. Thanks to this feature of C++ templates.

The value of a is 4

The value of b is 6.4

The value of c is c

The value of a is 1.6

The value of b is o

The value of c is c

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

**C++ Function Templates & Function Templates with Parameters | C++ Tutorials for Beginners #67**

In this tutorial, we are wishing to learn how a function template works. Prior to this video, we have only talked about a class template and its functionalities. In class template we used to have template parameters which we, very often, addressed as a variable for our data types. We have also declared a class template similar to what shown here below :

template <class T1 = int, class T2 = float>

Copy

Today, we’ll be interested in knowing what a function template does. So. let’s get ourselves on our editors.

Suppose we want to have a function which calculates the average of two integers. So, this must be very easy for you to formulate. Look for the snippet below.

1. We have declared a float function named funcAverage which will have two integers as its parameters, a and b.
2. We stored its average in a float variable avg and returned the same to the main.
3. Later we called this function by value, and stored the returned float in a float variable a and printed the same.
4. So this was the small effort we had to make to get a function which calculates the average of two integers.

#include<iostream>

using namespace std;

float funcAverage(int a, int 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 output of the above program is :

The average of these numbers is 3.500000

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

Copy

But the effort we made here defining a single function for two integers increases several folds when we demand for a similar function for two floats, or one float and one integer or many more data type combinations. We just cannot repeat the procedure and violate our DRY rule. We’ll use function templates very similar to what we did when we had to avoid defining more classes.

See what are the subtle changes we had to make, to make this function generic.

We’ll first declare a template with two data type parameters T1 and T2. And replace the data types we mentioned in the function with them. And that’s it. Our function has become general for all sorts of data types. Refer to the snippet below.

template<class T1, class T2>

float funcAverage(T1 a, T2 b){

float avg= (a+b)/2.0;

return avg;

}

Copy

Let’s call this function by passing into it two sorts of data types combination, first, two integers and then one integer and one float. And see if the outputs are correct.

int main(){

float a;

a = funcAverage(5,2);

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

return 0;

}

Copy

**Code snippet: Calling the function by passing two integers**

The average of these numbers is 3.500000

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

Copy

int main(){

float a;

a = funcAverage(5,2.8);

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

return 0;

}

Copy

**Code snippet: Calling the function by passing one integer and one float**

The average of these numbers is 3.900000

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

Copy

And a general swap function named swapp for those variety of data types we have, would look something like the one below:

template <class T>

void swapp(T &a, T &b)

{

T temp = a;

a = b;

b = temp;

}

Copy

So, this is how we utilize this powerful tool to avoid writing such overloaded codes. And this was all about function templates with single or multiple parameters. We covered them all in this tutorial.

**Member Function Templates & Overloading Template Functions in C++ | C++ Tutorials for Beginners #68**

So, since we have finished learning about the two template categories, we can now swiftly dive deep into if it's possible for a template function to get overloaded, and if yes, then how.

Before starting to know what an overloaded template function is, we’ll learn how to declare a template function outside a using the scope resolution operator, ‘::’.

First, we’ll revise how to write a function inside the class by just following the snippet given below.

1. We’ll declare a template, then a class named Harry.
2. We’ll then define a variable *data* inside that class with variable data type T.
3. We then make a constructor feeding the value received from the main to data.
4. And then, we’ll write the function, *display* and write its code.

This was an unchallenging task. But when we need the function to be declared outside the class, we follow the code snippet 2.

template <class T>

class Harry

{

public:

T data;

Harry(T a)

{

data = a;

}

void display()

{

cout << data;

}

};

Copy

**Code Snippet 1: Writing function inside the class**

Here, we first write the function declaration in the class itself. Then move to the outside and use the scope resolution operator before the function and after the name of the class Harry along with the data type T. We must specify the function data type, which is void here. And it must be preceded by the template declaration for class T.

And write the display code inside the function and this will behave as expected. See the output below the snippet.

template <class T>

class Harry

{

public:

T data;

Harry(T a)

{

data = a;

}

void display();

};

template <class T>

void Harry<T> :: display(){

cout<<data;

}

Copy

**Code Snippet 2: Writing function outside the class**

So to check if it's working all fine, we’ll call this function from the main.

int main()

{

Harry<int> h(5.7);

cout << h.data << endl;

h.display();

return 0;

}

Copy

**Code Snippet 3: Calling the function from the main**

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;

}

Copy

**Code Snippet 4: Overloading the template function**

And now when we call the function func, we’ll be interested to know which one among the two it calls. So here since we’ve entered a value with an integer parameter, it finds its exact match in the overloading and calls that itself. That is, it gives its exact match the highest priority. Refer to the output below the snippet:

int main()

{

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

return 0;

}

Copy

**Code Snippet 5: Calling function func from the main**

And the output is,

I am first func() 4

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

Copy

If we hadn’t created the first function with int data type,  the call would have gone to the templatised func only because a template function is an exact match for every kind of data type.

**The C++ Standard Template Library (STL) | C++ Tutorials for Beginners #69**

We have been waiting so long to start this, but creating a base is as important as any other phase. So, today we’ll be starting the most awaited topic, the STL( Standard Template Library).

There is a reason why I’ve been saying that this topic is a must for all the competitive programmers out there,so let’s deal with that first.

**Why is this important for competitive programmers?**

1. Competitive programming is a part of various environments, be it job interviews, coding contests and all, and if you’re in one of those environments, you’ll be given limited time to code your program.
2. So, suppose you want in your program, a resizable array, or sort an array or any other data structure. or search for some element in your container.
3. You will always try to code a function which will execute the above mentioned things, and end up losing a great amount of time. But here is when you will use STL.

An STL is a library of generic functions and classes which saves you time and energy which you would have spent constructing for your use. This helps you reuse these well tested classes and functions umpteenth number of times according to your own convenience.

To put this simply, STL is used because it is not a good idea to reinvent something which is already built and can be used to innovate things further. Suppose you go to a company who builds cars, they will not ask you to start from scratch, but to start from where it is left. This is the basic idea behind using STL.

**COMPONENTS OF STL:**

We have three components in STL:

1. Containers
2. Algorithm
3. Iterators

Let’s deal with them individually;

**Containers:**

Container is an object which stores data. We have different containers having their own benefits. These are the implemented template classes for our use, which can be used just by including this library. You can even customise these template classes.

**Algorithms:**

Algorithms are a set of instructions which manipulates the input data to arrive at some desired result. In STL, we have already written algorithms, for example, to sort some data structure, or search some element in an array. These algorithms use template functions.

**Iterators:**

Iterators are objects which refer to an element in a container. And we handle them very much similarly to a pointer. Their basic job is to connect algorithms to the container and play a vital role in manipulation of the data.

I’ll give you a quick illustration of how they work combinedly.

A screenshot of a computer

Description automatically generated with low confidence

**Figure 1: Illustration of how these three components work together**

Suppose we have a container of integers, and we want to sort them in ascending order. We will have pointers which will help moving elements to places by pointing to it, following a well-constructed algorithm. So, here a container gets sorted by following an algorithm by the use of pointers. This is how they work in accordance with each other.

**Containers in C++ STL | C++ Tutorials for Beginners #70**

In the last tutorial, we had briefed about the three components of STL, namely,

**Containers**, objects which store data, **Algorithms,**set of procedures to process data, and **Iterators,**objects which point to some element in a container. Today, in this tutorial, we will be interested in discussing more about containers**.**

Containers are themselves of three types:

1. Sequence Containers
2. Associative Containers
3. Derived Containers

When we talked about containers, we said containers are objects which store data, but what are its three types all about? We’ll discuss that too.

* **Sequence Containers**

A **sequence container**stores that data in a linear fashion. Refer to the illustration below to understand what storing something in a linear fashion means.

A picture containing application

Description automatically generated

**Figure 1: Elements stored in a linear fashion**

Sequence containers include **Vector, List, Dequeue etc.**These are some of the most used sequence containers.

* **Associative Containers**

An **associative container**is designed in such a way that enhances the accessing of some element in that container. It is very much used when the user wants to fastly reach some element. Some of these containers are, **Set, Multiset, Map, Multimap etc.**They store their data in a tree-like structure.

Diagram

Description automatically generated

* **Derived Containers**

As the name suggests, these containers are derived from either the sequence or the associative containers. They often provide you with some better methods to deal with your data. They deal with real life modelling. Some examples of derived containers are **Stack, Queue, Priority Queue, etc.**The following illustration give you the idea of how a stack works.

Diagram, table

Description automatically generated with medium confidence

**Figure 3: A stack, works on the first in first out [FIFO] method**

Now since we have got the basic idea of all the three types of containers, a question which might arise is **when to use which**.  So, let’s deal with that,

In sequence containers, we have **Vectors**, which has following properties:

1. Faster random access to elements in comparison to array
2. Slower insertion and deletion at some random position, except at the end.
3. Faster insertion at the end.

In **Lists,**we have,

1. Random accessing elements is too slow, because every element is traversed using pointers.
2. Insertion and deletion at any position is relatively faster, because they only use pointers, which can easily be manipulated.

In associative containers, every operation except random access is faster in comparison to any other containers, be it inserting or deleting any element.

In associative containers, we cannot specifically tell which operation is faster or slower, we’ll have to inspect every data structure separately, and to get a clearer picture of all of these, you can access my Data Structure course : [Data Structures and Algorithms Course in Hindi](https://youtube.com/playlist?list=PLu0W_9lII9ahIappRPN0MCAgtOu3lQjQi)

For now, I'd like to hold on to our topic STL, and get you a strong hold on this too. In the coming videos, we’ll deal with our vectors, list, dequeues, set, multiset, maps, stack and much more. Just bear with me.

**Vector In C++ STL | C++ Tutorials for Beginners #71**

In this video, we’ll cover the Vectors in C++ STL. This is the tutorial we all were waiting for. Enough of the theory part. We’ll go into our editors and code. So, to start, we’ll have to include the header file <vector>. And the syntax we use to define a vector is:

vector<data\_type> vector\_name;

Copy

**Code Snippet 1: Syntax of declaring a vector**

And suppose we want to have a vector of integers; the following program would do the needful:

#include<iostream>

#include<vector>

int main(){

vector<int> vec1;

return 0;

}

Copy

**Code Snippet 2: Declaring a vector of integers**

One benefit of using vectors, is that we can insert as many elements we want in a vector, without having to put some size parameter as in an array. In an array of 10 elements, for adding the 11th one, we’ll have to make an array again.

Vectors provide certain methods to be used to access and utilise the elements of a vector, first one being, the push\_back method. To access all the methods and member functions in detail, you can visit this site , [std::vector - C++ Reference](https://www.cplusplus.com/reference/vector/vector/). This will be very handy and useful to you.  I’ll show you how some of them work in a program. Refer to the code snippet 3.

* **push\_back() and size():**

1. First of all, don't forget to include the header file, <vector>.
2. Vectors have a method, push\_back(), to insert elements in it from the rear end.
3. We’ll define a variable, size, to store the size of the vector.
4. We’ll then run a loop of size length, to receive the user input and push them back in the vector vec1.
5. We’ll then call the display function.
6. We want to have a display function to display the contents of the vector. And pass reference of vec1 to the function.
7. We have another method size() which returns the size of the vector. We’ll use this to traverse through all the elements and print them.
8. So, this is how a vector gets used.

#include<iostream>

#include<vector>

using namespace std;

void display(vector<int> &v){

for (int i = 0; i < v.size(); i++)

{

cout<<v[i]<<" ";

}

cout<<endl;

}

int main(){

vector<int> vec1;

int element, size;

cout<<"Enter the size of your vector"<<endl;

cin>>size;

for (int i = 0; i < size; i++)

{

cout<<"Enter an element to add to this vector: ";

cin>>element;

vec1.push\_back(element);

}

display(vec1);

return 0;

}

Copy

**Code Snippet 3: A program to demonstrate the use of push\_back and size methods**

The output of the above program:

Enter the size of your vector

3

Enter an element to add to this vector: 5

Enter an element to add to this vector: 3

Enter an element to add to this vector: 7

5 3 7

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

Copy

**Figure 1: Output of the above program**

Similarly, we can even build float vectors, and we can even templatise the display function.

* **pop\_back():**

This method of vectors, deletes the last element of the vector. Refer to the code snippet and the following output below.

display(vec1);

vec1.pop\_back();

display(vec1);

Copy

**Code Snippet 4: Using pop\_back in a vector**

So, now you can see how this method deleted the last element 7 from the vector.

Enter the size of your vector

3

Enter an element to add to this vector: 5

Enter an element to add to this vector: 3

Enter an element to add to this vector: 7

5 3 7

5 3

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

Copy

**Figure 2: Output of the above program**

* **Insert (**iterator, element to insert**):**

This method of vectors inserts an element to the position the iterator is pointing to. Now how to evoke that iterator? Refer to the snippet and the output below:

We can generate an iterator using the scope resolution iterator by the following syntax:

vector<int> :: iterator iter = vec1.begin();

Copy

**Code Snippet 5: Declaring a vector iterator**

Using **begin ()**points the iterator to the starting of the vector. We can now increment the pointer according to our choice and insert any element at that position.

display(vec1);

vector<int> :: iterator iter = vec1.begin();

vec1.insert(iter,566);

display(vec1);

Copy

**Code Snippet 6: Demonstrating an insert method**

The output of the above program is:

Enter the size of your vector

3

Enter an element to add to this vector: 5

Enter an element to add to this vector: 3

Enter an element to add to this vector: 7

5 3 7

566 5 3 7

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

Copy

**Figure 3: Output of the above program**

Similarly,**v.at(i)** can be used instead of **v[i]**. They will work the same.

We have different ways to declare a vector. I’ll list some of them through the snippet below.

1. First one is a vector with no length and elements specified.
2. Second one is a vector of length 4 and no elements.
3. Third one is a vector made from the second one.
4. And last one, is a vector with length 6 and all the elements being 3.

vector<int> vec1; //zero length integer vector

vector<char> vec2(4); //4-element character vector

vector<char> vec3(vec2);//4-element character vector from vec2

vector<int> vec4(6,3); //6-element vector of 3s

Copy

**Code Snippet 7: Demonstrating different ways to declare a vector**

**List In C++ STL | C++ Tutorials for Beginners #72**

Before this tutorial, we covered templates, STL, and the last video was an efficient introduction to the vectors. Today, we’ll learn about Lists in C++ STL.

A List is a bi-directional linear storage of elements. Few key features as to why a list should be used is,

1. It gives faster insertion and deletion operations.
2. Its access to random elements is slow.

**What makes a list different from an array?**

An array stores the elements in a contiguous manner in which inserting some element calls for a shift of other elements, which is time taking. But in a list, we can simply change the address the pointer is pointing to. I’ll show you how these work via an illustration.

Text

Description automatically generated with medium confidence

Diagram

Description automatically generated

Let's move on to our editors and write some code using lists and its methods.

**Understanding code snippet 2:**

* Before using lists, we must include the header file <list>.
* Using a simple program, we'll iterate through the list and display its contents.
* As we did for vectors, first define a list list1.
* And push\_back a few elements, and pass the list to a display function via reference.
* Due to the fact that a list element cannot be directly accessed by its index, we must traverse through each element and print them.
* We define a list iterator using this syntax:

list<int> :: iterator it;

Copy

**Code Snippet 1: Syntax for defining a list iterator**

* We use two methods, **begin()** and **end()** to define the starting and the end of the loop. **end()** returns the pointer next to the last element.
* We dereference the list iterator, using \* to print the element at that index.

#include<iostream>

#include<list>

using namespace std;

void display(list<int> &lst){

list<int> :: iterator it;

for (it = lst.begin(); it != lst.end(); it++)

{

cout<<\*it<<" ";

}

}

int main(){

list<int> list1; //empty list of 0 length

list1.push\_back(5);

list1.push\_back(7);

list1.push\_back(1);

list1.push\_back(9);

list1.push\_back(12);

display(list1);

return 0;

}

Copy

**Code Snippet 2: A program using list**

5 7 1 9 12

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

Copy

**Figure 3: Output of the above program**

We can also enter elements in a list using the iterator and its dereferencer. See the snippet below.

int main(){

list<int> list2(3); //empty list of length 3

list<int> :: iterator it = list2.begin();

\*it = 45;

it++;

\*it = 6;

it++;

\*it = 9;

it++;

display(list2);

return 0;

}

Copy

**Code Snippet 3: Inserting in list using its iterator**

45 6 9

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

Copy

**Figure 4: Output of the above program**

* **Using pop\_back() and pop\_front():**

We can use pop\_back() to delete one element from the back of the list everytime we call this method and pop\_front() to delete elements from the front. These commands decrease the size of the list by 1. Let me show you how these work by using them for list1 we made.

list1.pop\_back();

display(list1);

list1.pop\_front();

display(list1);

Copy

**Code Snippet 4: Using pop\_back and pop\_front in list**

The output of the above program is:

5 7 1 9

7 1 9

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

Copy

**Figure 5: Output of the above program**

* **Using remove():**

We can remove an element from a list by passing it in the list remove method. It will delete all the occurrences of that element. The remove method receives one value as a parameter and removes all the elements which match this parameter. Refer to the use of remove in the below snippet.

int main(){

list<int> list1; //empty list of 0 length

list1.push\_back(5);

list1.push\_back(7);

list1.push\_back(1);

list1.push\_back(9);

list1.push\_back(9);

list1.push\_back(12);

list1.remove(9);

display(list1);

return 0;

}

Copy

**Code Snippet 5: Deleting elements in list using remove()**

The output of the above program is:

5 7 1 12

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

Copy

**Figure 6: Output of the above program**

* **Using sort():**

We can sort a list in ascending order using its sort method. Look for the demo below.

display(list1);

list1.sort();

display(list1);

Copy

**Code Snippet 6: Sorting elements in list using sort()**

5 7 1 9 12

1 5 7 9 12

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

Copy

**Figure 7: Output of the above program**

I consider this much to be enough for lists. There are still a lot of them, but you will require no more, and even if you feel like exploring more, move onto [std::list - C++ Reference](https://www.cplusplus.com/reference/list/list/) and read about all the lists methods. This was all from my side.

**Map In C++ STL | C++ Tutorials for Beginners #73**

So far, we have learned about vectors and lists in C++ STL, and today we will be learning about maps in C++ STL. It is important to clarify that whatever I have taught and whatever I will be teaching in the coming tutorials about STL isn’t everything. And it is definitely not all. These are just the most important STL containers we will use. You have already seen how to explore more about STL from [C++ - Containers](https://www.cplusplus.com/reference/stl/).

We will now discuss maps, and because it is impractical to have every method on our fingers, I’d ask you all to also refer to the following website [std::map - C++ Reference](https://www.cplusplus.com/reference/map/map/)

A map in C++ STL is an associative container which stores key value pairs. To elaborate, a map stores a key of some data type and its corresponding values of some data type. For example: a teacher wants to store the marks of students which in future can be accessed by their names. Here, keys are the student names, and their marks are the corresponding values. Refer to the illustration below:

Diagram, table

Description automatically generated

We can now shift to our editors and see how maps can be used in C++. Don’t forget to include the header file <map>.

The syntax for declaring a map is:

map <data\_type\_of\_key, data\_type\_of\_value> variable\_name;

Copy

**Code Snippet 1: Syntax for declaring a map**

And we can now write the program for storing the key value pairs of students' names and students’ marks keeping in mind the illustration above. Refer to the snippet below.

**Understanding code snippet 2:**

1. Include the header file map and string( if using string methods).
2. Let's create a map in which the key is a string (names) and the values are integers (marks), and we'll call it marksMap.
3. And to assign some key a value, we use the index method. Here the index of a map element will be the students’ name and the value will be the marks.
4. Make some 4-5 elements.
5. Identify the iterator of this map by using the scope resolution operator.
6. Loop through the map elements using two map methods; **begin()** to point at the beginning of the map, and **end()** to point next to the last element of the map.
7. While we loop through the map, we use the dereference operator \* to fetch the element present where the pointer is pointing to. And since a map stores element in a key value pair, we can use its first and second method to access the keys and the values respectively. **.first**accesses the first value of a pair that is our map key here, and **.second**accesses the second value of the pair that is our map values here.
8. There is one thing to keep in mind: Maps always sort these pairs by the key elements. You can review the output of the following snippet to see how these pairs are sorted.

#include<iostream>

#include<map>

#include<string>

using namespace std;

int main(){

// Map is an associative array

map<string, int> marksMap;

marksMap["Atul"] = 58;

marksMap["Rohit"] = 57;

marksMap["Kishlay"] = 78;

marksMap["Aditya"] = 65;

marksMap["Sachin"] = 53;

map<string,int> :: iterator iter;

for (iter = marksMap.begin(); iter != marksMap.end(); iter++)

{

cout<<(\*iter).first<<" "<<(\*iter).second<<"\n";

}

return 0;

}

Copy

**Code Snippet 2: A program to store names and marks using map**

As you can see, the map pairs got sorted according to its key.

Aditya 65

Atul 58

Kishlay 78

Rohit 57

Sachin 53

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

Copy

**Figure 2: Output of the above program**

We have one more method to insert elements in a map. We can use **.insert()**

Syntax for using .insert is:

marksMap.insert({pair\_1,pair\_2......pair\_n});

Copy

**Code Snippet 3: Syntax for inserting pairs in map**

We will insert some elements into our map in snippet 2 by using the insert method.

marksMap.insert( { {"Rohan", 89}, {"Akshat", 46} } );

Copy

**Code Snippet 4:  Program to insert two pairs in a map.**

We can see the output to check if the above two pairs got inserted into the map or not.

Aditya 65

Akshat 46

Atul 58

Kishlay 78

Rohan 89

Rohit 57

Sachin 53

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

Copy

**Figure 3: Output of the above program**

So, yes, it worked. And the output was correct.

And I’d ask you all to explore more of these methods from the links I gave you above and start writing codes using them. This is the best way to learn. For example, you can use the **size()**method to get the size of the map container , **empty()**method to check if the map container is empty or not, and it returns a boolean. Therefore, this shouldn't be a big deal for you.

**Function Objects (Functors) In C++ STL | C++ Tutorials for Beginners #74**

In the last tutorial we completed learning about some of the most commonly used containers, vector, list, map and their methods. Today we’ll start with function objects in C++ STL.

**What is a function object?**

A function object is a function wrapped in a class so that it is available as an object.

That is, we can then use a function as an object. The question that might have been raised in your mind would be,**why to substitute a function with an object**? The answer is to make them all usable in an Object-Oriented Programming paradigm. Now what does that mean? We’ll try decoding the purpose of using functions as an object via a program. So, hold onto your editors.

**Understanding code snippet 1:**

* Be sure to include the header file < functional> before you do anything else.
* And let’s create an array of some 6 elements.
* Suppose we want to sort this array in ascending order. So we’ll include a header file <algorithm> and write the syntax of the sort object which is,

sort(address of first element, address of last element);

Copy

**Code Snippet 1: Syntax for sort algorithm**

* And let’s just sort from the beginning to the 5th element.
* And run a loop to see the resultant array.

#include<iostream>

#include<functional>

#include<algorithm>

using namespace std;

int main(){

// Function Objects (Functor) : A function wrapped in a class so that it is available like an object

int arr[] = {1, 73, 4, 2, 54, 7};

sort(arr,arr+5);

for (int i = 0; i < 6; i++)

{

cout<<arr[i]<<endl;

}

return 0;

}

Copy

**Code Snippet 2: Program to sort an array in ascending order**

Output of the above program is given below.  And you’ll notice that the last element remained untouched.

1

2

4

54

73

7

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

Copy

**Figure 1: Output of the above program**

But what if we wanted to sort the same array in descending order, since the sort function can default sort in ascending order only? So, here comes our saviour, **functional objects**. Our sort function also takes a third parameter which is a functor ( functional object).

Let’s see how they work via the snippet below:

* Among all the different functors we have, the one to help this sort function to sort the array in descending order, is the **greater<int>()**.

sort( arr, arr+6, greater< int >( ));

Copy

**Code Snippet 1: Syntax for using a functor in an algorithm**

* And that’s it. Our array will now get sorted in descending order.

See the output after the above changes we made:

73

54

7

4

2

1

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

Copy

**Figure 1: Output of the above program after using a functor greater<int>()**

It would be unnecessarily lengthy to review all the functors, as my role was to introduce them to you and show you how they are used.

In addition, I invite you all to explore the other function objects on the site [Function objects](https://en.cppreference.com/w/cpp/utility/functional#:~:text=A%20function%20object%20is%20any,manipulation%20of%20new%20function%20objects). You should go through them lightly because a lot of them would be overwhelming to you as a beginner. We use most of these functors for our STL algorithms, so you might want to check all those algorithms on [Functions in <algorithm> - C++ Reference](https://www.cplusplus.com/reference/algorithm). You can go through them at your own pace. Incorporate them into your programs. Take advantage of them and use them how you see fit.