**Experiment No: 6 Date:- 24-11-2020**

**AIM: To study Templates in C++**

**THEORY:**

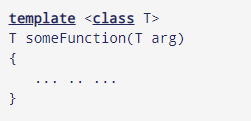
Templates are powerful features of C++ which allows you to write generic programs. In simple terms, you can create a single function or a class to work with different data types using templates. Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs.

The concept of templates can be used in two different ways:

* Function Templates
* Class Templates

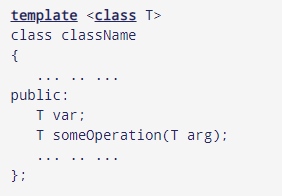
**Function Templates**

A function template works in a similar to a normal function, with one key difference. A single function template can work with different data types at once but, a single normal function can only work with one set of data types. Normally, if you need to perform identical operations on two or more types of data, you use function overloading to create two functions with the required function declaration. However, a better approach would be to use function templates because you can perform the same task writing less and maintainable code.



**Class Templates**

Like function templates, you can also create class templates for generic class operations. Sometimes, you need a class implementation that is same for all classes, only the data types used are different. Normally, you would need to create a different class for each data type OR create different member variables and functions within a single class. This will unnecessarily bloat your code base and will be hard to maintain, as a change is one class/function should be performed on all classes/functions. However, class templates make it easy to reuse the same code for all data types.



**A] Write a C++ program to implement a function template to swap two elements**

**#include<iostream>**

**using std::cout ;**

**template<class T>**

**void swap(T &a, T &b)**

**{**

**T temp=a;**

**a=b;**

**b=temp;**

**}**

**int main()**

**{**

**int a=4,b=90;**

**float c=1.4,d=3.95;**

**cout<<"Before Swap:";**

**cout<<"\na="<<a<<"\tb="<<b;**

**cout<<"\nc="<<c<<"\td="<<d;**

**swap(a,b);**

**swap(c,d);**

**cout<<"\n\nAfter Swap:";**

**cout<<"\na="<<a<<"\tb="<<b;**

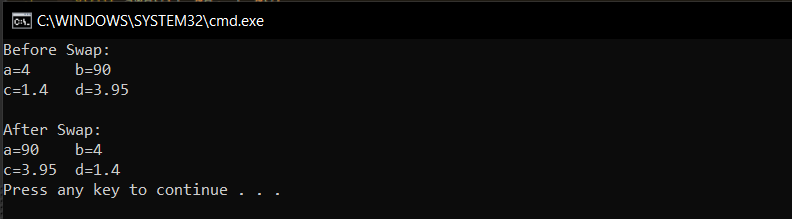
**cout<<"\nc="<<c<<"\td="<<d;**

**cout<<"\n";**

**return 0;**

**}**

**Output :**

****

**B] Write a C++ program to create a class template to represent a generic vector. Include the member functions to perform the following tasks**

1. **Create the vector**
2. **To modify the value of a given element**
3. **To display the vector elements**

#include<iostream>

using namespace std;

template<class T>

class vector

{

T \*v;

int size;

public:

vector(T \*a, int s)

{

v = new T[size=s];

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

v[i]=a[i];

}

void display()

{

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

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

cout<<endl;

}

void modify(T x, int s)

{ s=s-1;

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

if(i==s)

v[i]=x;

}

};

int main()

{

int a[3]={5,8,10};

vector<int>t(a,3);

cout<<"The vector is as follows"<<endl;

t.display();

int i,x;

cout<<"Enter the position of the element to be modified and enter new element"<<endl;

cin>>i>>x;

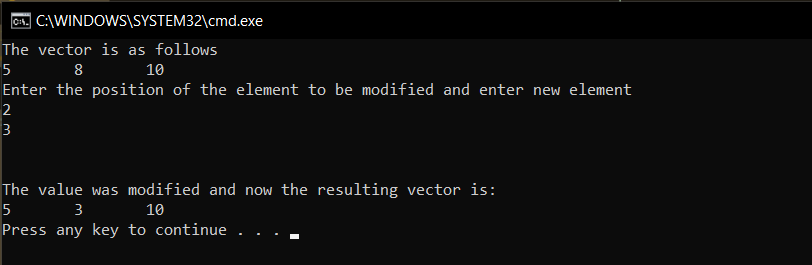
t.modify(x,i);

cout<<"\n\nThe value was modified and now the resulting vector is:"<<endl;

t.display();

return 0;

}



**CONCLUSION:**

All programs were implemented and run successfully, and desired output was obtained.

* A template is a simple and yet very powerful tool in C++. The simple idea is to pass data type as a parameter so that we don’t need to write the same code for different data types.
* For example, an IT firm may need a sort function for different data types. Rather than writing and maintaining the multiple codes for different data types, we can write one sort() and pass data type as a parameter.