**NOTES**

1. Templates are used when we want to avoid duplication in coding. Like we make one class for one data type and if we make same class for another data type but with similar functionality or we make functions overloading, then code maintenance would be difficult. So it would be easier to maintain as we have updates and we will have to update both.
2. Consider the following:

#include <iostream>

using namespace std;

class intArray

{

public:

    int array[10];

    void fill(int value){

        for (int i = 0; i < 10; i++){

            array[i] = value;

        }

    }

    int at(int index){

        return array[index];

    }

};

class stringArray

{

public:

    string array[10];

    void fill(int value){

        for (int i = 0; i < 10; i++){

            array[i] = value;

        }

    }

    string at(int index){

        return array[index];

    }

};

int main(){

 intArray arr1;

 arr1.fill(4);

 cout << arr1.at(5) << endl;

stringArray arr2;

 arr2.fill("Hello");

  cout << arr2.at(6) << endl;

    return 0;

}

1. Class templates allow us to define Generic class and functions and avoid duplications.
2. This is how we can do it

template <typename T>

class Array

{

public:

    T array[10];

    void fill(T value){

        for (int i = 0; i < 11; i++){

            array[i] = value;

        }

    }

    T at(int index){

        return array[index];

    }

};

* 1. And we can call the class like this

 Array<int> arr3;

 arr3.fill(3);

 cout << arr3.at(7) << endl;

* 1. Now we can also update our template class with more parameters.
  2. Like this

template <typename T, int length>

class Array

{

public:

    T array[length];

    void fill(T value){

        for (int i = 0; i < length; i++){

            array[i] = value;

        }

    }

    T at(int index){

        return array[index];

    }

};

* 1. And create instances like this

Array<int, 8> arr3;

 arr3.fill(3);

 cout << arr3.at(7) << endl;

* 1. It is important to remember that the length expression must be constant and well defined not a variable. It will give errors otherwise.

Can also show some code implementation

**Function Templates**

**// C++ Program to demonstrate**

**// Use of template**

**#include <iostream>**

**using namespace std;**

**// One function works for all data types. This would work**

**// even for user defined types if operator '>' is overloaded**

**template <typename T> T myMax(T x, T y)**

**{**

**return (x > y) ? x : y;**

**}**

**int main()**

**{**

**// Call myMax for int**

**cout << myMax<int>(3, 7) << endl;**

**// call myMax for double**

**cout << myMax<double>(3.0, 7.0) << endl;**

**// call myMax for char**

**cout << myMax<char>('g', 'e') << endl;**

**return 0;**

**}**

Example of Bubble Sort

// C++ Program to implement

// Bubble sort

**// using template function**

**#include <iostream>**

**using namespace std;**

**// A template function to implement bubble sort.**

**// We can use this for any data type that supports**

**// comparison operator < and swap works for it.**

**template <class T> void bubbleSort(T a[], int n)**

**{**

**for (int i = 0; i < n - 1; i++)**

**for (int j = n - 1; i < j; j--)**

**if (a[j] < a[j - 1])**

**swap(a[j], a[j - 1]);**

**}**

**// Driver Code**

**int main()**

**{**

**int a[5] = { 10, 50, 30, 40, 20 };**

**int n = sizeof(a) / sizeof(a[0]);**

**// calls template function**

**bubbleSort<int>(a, n);**

**cout << " Sorted array : ";**

**for (int i = 0; i < n; i++)**

**cout << a[i] << " ";**

**cout << endl;**

**return 0;**

**}**