C++ Slides - 7

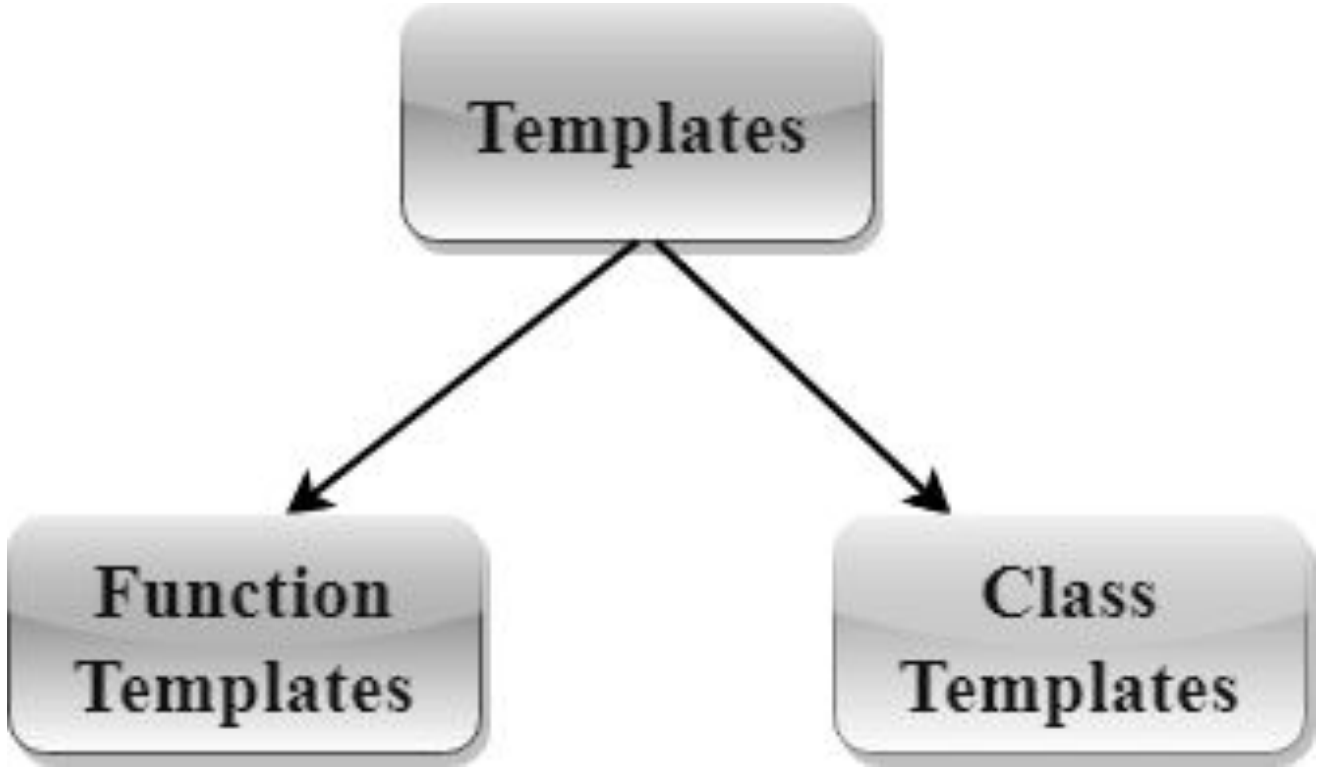
Templates: Need of template, Function templates, Function template with non-type parameter, Overloading function templates, Class templates, Class template with non-type parameter.

Need of template - motivation

- To reduce code *duplication* when supporting numerous data types

- int MyMax(int x, int y){ return x>y?x:y; }
- float MyMax(float x, float y){ return x>y?x:y; }
- char MyMax(char x, char y){ return x>y?x:y; }

- T MyMax(T x, T y){ return x>y?x:y;}



We can write a generic code for a function e.g. add() for integers, double, float etc.

We can write a generic code for a class, to manipulate group of member variables & functions e.g. linked list of strings, integers etc.

How to use template – an example

```
#include <iostream>
using namespace std;
template <class T>
T myMax(T x, T y){ return x>y?x:y;}
int main(){
cout<<myMax(10,20)<<endl;
cout<<myMax('a','z')<<endl;
cout<<myMax(-2.5,7.7)<<endl;
}
```

Function Templates(Generics) - *definition*

- We write a generic function that can be used for different data types. Syntax of an example function -

```
template <class T>
T myMax(T x, T y){ return x>y?x:y;}
```

- Other examples could be sort(), max(), min(), printArray(), show() etc.

Exercise – Make the generic template of -

```
void bubbleSort(int 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]);
}
```

Answer:

```
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]);
}
```

Exercise - Write a template for

```
int main(){
show(100,"hello hello");
show('k',1500);
show(1.23,2987);
}
```

Answer:

```
template <class T1, class T2>
void show(T1 a, T2 b){
cout<<a<<“, “<<b<<endl;
}
```

Function template with non-type parameter

- Non-type parameter is not a type (datatype) but a value e.g. 100
- They are used to initialize a class or to specify the sizes of class members

- template <class T, int size> // size is the non-type parameter

Function template with non-type parameter

```
#include<iostream>
using namespace std;
template <class T, int size>
void show(T a){cout<<a<<", "<<size;}
int main(){
show <char,10> ('c');
}
```

Overloading function templates

```
template <class T1, class T2>
void show(T1 a, T2 b){cout<<a<<", "<<b<<endl;}
void show(int a, int b){cout<<”For integer cases";}
int main(){
show(100,"hello hello");
show(3,3);
}
```

```
// Template class example
template <class T>
class Test {
T var;
public:
Test (T i) {var=i;}
T divideBy2 () {return var/2;}
};
int main(){
Test <int> t1(50);
Test <double> t2(-10.20);
cout<<t1.divideBy2()<<" "<<t2.divideBy2()<<endl;
}
```

Defining function outside the template class - *complicated*

```
template <class T>
class Test {
T var;
public:
Test (T i) {var=i;}
T divideBy2 ();
};
template <class T>
T Test<T> :: divideBy2(){return var/2;}      // complicated
```

```
// Class template with non-type parameter
template <class T, int n>
class Test {
T var;
public:
Test () {var = n; cout<<"n = "<<n<<endl;}
T divideBy2 () {return var/2;}
};
int main(){
Test <int,10> t1;
Test <double,20> t2;
cout<<t1.divideBy2()<<" "<<t2.divideBy2()<<endl;
}
```

Output

n = 10

n = 20

5 10