Practical Lecture : Templates Day 1

Quick Recap

Let’s take a quick recap of previous lecture –

- Basics of exception handling
- Exception handling mechanism
- Throwing mechanism
- Catching mechanism
- Rethrowing an exception

Today’s Agenda

Today we are going to cover –

- Introduction to templates
- Function template
- class template

Let’s Get Started-

Introdution

Note: Encourage students to answers the following questions.

1. What are templates?

Ans: In general these are standard forms .

2. Have you all used a templates in your life?

Ans: Yes. We all have used templates somewhere in our life

3. Give example.

Ans: Suppose you go to bank to deposit or withdraw money. In earlier times, we had to fill the withdrawal or deposit form where we put necessary information like name, account number, amount, etc. Imagine what will happen if the form is not available? Everyone will try to provide the information in his / her own format. This form is called template.

Other examples of templates could be exam form, admission form, etc.

Why templates

Let us understand this with the help of example.

```
#include<iostream>
using namespace std;
int Max( int a, int b){
        return a>b ? a :b;
}
int main(){
        cout<< "Max of two numbers : " <<Max(4,5);
        return 0;
}
```

The above program gives max to two integers. What if we want to use it for doubles?

Why templates

For doubles, we have to overload the function.

```
#include<iostream>
using namespace std;
int Max( int a, int b){
	return a>b ? a :b;
}
double Max( double a, double b){
	return a>b ? a :b;
}
int main(){
	cout<<Max(4,5)<<endl;
	cout<<Max(4.5, 3.2)<<endl;
	return 0;
}
```

Why Templates

Similarly if we want to find out max of two strings , we have to modify the function to handle strings.

So for any new data type we want to use the same function, we have to overload the function with this new data type.

Instead we can make life simple by writing code in a way that is independent of any particular type.

This concept is called as template. 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.

Templates

A template is a blueprint or formula for creating a generic class or a function

C++ supports two types of templates: **1. Function 2. Class**

Template is essential feature added recently to C++.

This new concept allows programmers to define generic classes and functions and thus provide support for generic programming.

Generic programming is an approach of C++ programming where generic types are used as parameters so that they can work for various cases of suitable data type and data structure.

The template is the basis for establishing the concept of generic programming, which entails writing code in a way that is independent of a particular type.

Function template

Practice question

```
#include <iostream>
using namespace std;
template <class X>  //additional line for template syntax
X Max (X a, X b) {  //X is any data type
//The above two lines could be written on same or different lines as shown below
//template <class X> X Max (X a, X b){
return a >b ? a:b;
}
int main(){
        cout<<Max(4,5)<<endl;
        cout<<Max(4.5, 3.2)<<endl;
        return 0;
}
```

Output:

5

4.5

Practice question

Explanation:
In addition to normal program, we have preceded the function definition with template <class T> which we always do.

X Max (X a, X b): is a function prototype where X could be any name you like. X works as placeholder written in <>. If you pass int as data type, X will be int. If you pass double as data type, X will be double.

syntax

A function template is also known as generic function.

Syntax:

```
template <class type> ret-type function_name(parameter list)
{
// body of function
}
```

template: is a keyword and is to be written in small case

class is a keyword which is madatory

Type is a placeholder which will be replaced with data type you pass from calling function.

Ret-type function_name (parameter list) : is a normal function prototype.

Practice question

What if I have different types of arguments in function call?

The above program can be modified with reference to the following syntax:

template <class type1, class type2> type1 or type2 function_name (type1 arg1, type2 arg2, ...)

Let us understand it better with the help of example.

Hint: Initially, Always write the program in usual way and then modify it to make it generic.

Practice question

```
#include <iostream>
using namespace std;
template <class T, class M> M Max(T a, M b)
{
	return a>b ? a :b;
}
int main()
{
	cout<<(3,4.6)<<endl;
	cout<<(4.5, 4)<<endl;
	return 0;
}
```

Output:

4.6

4

//second output is 4 as first argument is truncated from double to int as 4 .

Practice question

Write a function template to accept an array and its size and return sum of elements of an array.
Let us first write the simple program using function. Then convert to generic program.

```
#include <iostream>
using namespace std;
int Sum (int a[], int size)
{
	int s=0;
	for (int i=0; i<size;i++)
	{
		s=s+a[i];
	}
	return s;
}
```

Practice question

```
int main()
{
        int x[]={10,20,30,40,50};
        //double y[]={ 1.1,2.2,3.3};
        cout<<"Sum of int array="<<Sum(x ,5);
}
```

Output:

Sum of int array=150

Practice question

Let us now modify the above program using template

```
template <class T>T Sum (T a[], int size)
{
	T s=0;
	for (int i=0; i<size;i++)
	{
		s=s+a[i];
	}
	return s;
}
int main() {
	int x[]={10,20,30,40,50};
	double y[]={ 1.1,2.2,3.3};
	cout<<"Sum of int array="<<Sum(x ,5)<<endl;
	cout<<"Sum of double array= "<<Sum(y,3)<<endl;
}
```

Output:

Sum of int array=150

Sum of double array=6.6

Practice question

Let us now modify the above program using template

```
template <class T>T Sum (T a[], int size)
{
	T s=0;
	for (int i=0; i<size;i++)
	{
		s=s+a[i];
	}
	return s;
}
int main() {
	int x[]={10,20,30,40,50};
	double y[]={ 1.1,2.2,3.3};
	cout<<"Sum of int array="<<Sum(x ,5)<<endl;
	cout<<"Sum of double array= "<<Sum(y,3)<<endl;
}
```

Output:

Sum of int array=150

Sum of double array=6.6

How templates work

Templates are expanded at compiler time. This is like macros.
The difference is, the compiler does type checking before template expansion. The idea is simple, source code contains only function/class, but compiled code may contain multiple copies of same function/class.

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

int main()
{
   cout << myMax<int>(3, 7) << endl;
   cout << myMax<char>('g', 'e') << endl;
   return 0;
}
```

Compiler internally generates and adds below code

```
int myMax(int x, int y)
{
    return (x > y)? x: y;
}
```

Compiler internally generates and adds below code.

```
char myMax(char x, char y)
{
    return (x > y)? x: y;
}
```

Assignment

Create a template function for addition of two numbers. This function returns the sum of the numbers to calling function. Pass different types of data to it :
(int, int)
(double, double)
(int , double)
(double, int)

Observe the advantages of template with respect to function overloading.

Class template

Class template

Class Templates: Like function templates, class templates are useful when a class defines something that is independent of the data type.

Syntax:

```
template <class type>
class class-name
{
        …
        …
        …
 }
```

Here, type is the placeholder type name, which will be specified when a class is instantiated.

Practice question

Let is create a class with 2 data members and a constructor

```
class A  {
    int x;
    int y;
public:
    A() {    cout<<"Constructor Called"<<endl;   }
};

int main()  {
   A a;
   return 0;
}
```

Output: Constructor Called

Practice question

Let us modify it to display the sum.

```
class A
{
        int x;
        int y;
        public:    A()
        {
                x=10;       y=2;
                cout<<"Constructor Called"<<endl;
        }
        int display()
        {
                return x+y;
        }
};
```

```
int main()
 {
        A a;
        cout<<a.display();
        return 0;
}
```

Output:

Constructor Called

12

Practice question

Let us modify to make it generic class

```
Template <class T>
class A
{
        T x;
        T y;
        public:    A()
        {
                x=10;       y=2;
                cout<<"Constructor Called"<<endl;
        }
        T display()
        {
                return x+y;
        }
};
```

```
int main()
 {
        A <int> a;
        cout<<a.display();
        return 0;
}
```

Output:

Constructor Called

12

Practice question

Explanation:

- To make generic class, we precede class definition with Template <class T> .
- In class definition, whichever data type we want to make generic , we will replace it with T. Here int is replaced with T.
- T x; T y; // replaced int x;int y;
- Also int display() replaced with T display() as we want the function to return the int value of sum of the variables.
- In main(), usually we create objects using A a; (i.e. classname objectname). In this case, memory is allocated to objects depending on types of data members.
- But since we declared variables of type T, compiler does not know how much memory to allocate for object ‘a’ as T is unknown. Hence to give hint to compiler, we must create objects using following syntax:
- A <int> a; //will create objects and allocate memory for integers
- A <double > b; //will create objects and allocate memory for doubles.

Practice question

Let us modify it to have parameterized constructor

```
Template <class T>
class A
{
        T x;
        T y;
        public:    A(T m, T n)
        {
                x=m;        y=n;
                cout<<"Constructor Called"<<endl;
        }
        T display()
        {
                return x+y;
        }
};
```

```
int main()
 {
        A <int> a(2,3);
        cout<<a.display()<<endl;
        A <double> a(3.4, 5.7);
        cout<<a.display()<<endl;


        return 0;
}
```

Output:

Constructor Called

5

Constructor Called

Practice question

Let us modify it to handle two different data types

```
Template <class T, class W>
class A
{
        T x;
        W y;
        public:    A(T m, W n)      {
                x=m;       y=n;
                cout<<"Constructor Called"<<endl;
        }
        double display()    //always want double
irrespective of input
        {
                return x+y;
        }
};
```

```
int main()
 {
        A <int, double> a(2,3.4);
        cout<<a.display()<<endl;
        A <double,int> a(3.4, 5);
        cout<<a.display()<<endl;
        return 0;
}
```

Output:

Constructor Called

5.4

Constructor Called

Practice question

The only member function in the previous class template has been defined inline within the class declaration itself.
In case that we define a function member outside the declaration of the class template, we must always precede that definition with the template <...> prefix:
Let us see how the member function definition will be modified.

Practice question

Let us modify it to have the member function definition outside the class

```
Template <class T>
class Myclass
{
        T x;
        T y;
        public:    Myclass(T m, T n)
        {
                x=m;        y=n;
                cout<<"Constructor Called"<<endl;
        }
        T display() ;
};
```

```
template <class T>
T Myclass <T> :: display()
{
        return x+y;
}

int main()
 {
        Myclass <int> a(2,3);
        cout<<a.display()<<endl;
        return 0;
}
```

Output:

Constructor Called

Practice question

Explanation:

```
T Myclass :: display() //ideally this would have been the definition for generic member .
{
        return x+y;
}
template <class T>   //but highlighted below are the additional changes
T Myclass <T> :: display()
{
        return x+y;
}
```

Confused by so many T's? There are three T's in this declaration: The first one is the template parameter. The second T refers to the type returned by the function. And the third T (the one between angle brackets) is also a requirement: It specifies that this function's template parameter is also the class template parameter.

Assignment

Provide the definition for the function display() out the class.

```
Template <class T, class W>
class A
{
        T x;
        W y;
        public:    A(T m, W n)       {
                x=m;        y=n;
                cout<<"Constructor Called"<<endl;
        }
        double display() ;
};
// display()???
```

```
int main()
 {
        A <int, double> a(2,3.4);
        cout<<a.display()<<endl;
        A <double,int> a(3.4, 5);
        cout<<a.display()<<endl;
        return 0;
}
```

Output:

Constructor Called

5.4

Constructor Called

Practice question

Which of the following is true about templates.
1) Template is a feature of C++ that allows us to write one code for different data types.

2) We can write one function that can be used for all data types including user defined types.

3) We can write one class or struct that can be used for all data types including user defined types.

4) Template is an example of compile time polymorphism.

Options:

1. 1,2
2. 1,2,3
3. 1,2,4
4. 1,2,3,4

Practice question

Which of the following is true about templates.
1) Template is a feature of C++ that allows us to write one code for different data types.

2) We can write one function that can be used for all data types including user defined types.

3) We can write one class or struct that can be used for all data types including user defined types.

4) Template is an example of compile time polymorphism.

Options:

1. 1,2
2. 1,2,3
3. 1,2,4
4. 1,2,3,4

Practice question

What will be the output of the following program?

```
template <typename T>
void fun(const T&x)
{
   static int count = 0;
   cout << "x = " << x << " count = " << count << endl;
   ++count;
   return;
}
```

```
int main()
{
   fun<int> (1);
   cout << endl;
   fun<int>(1);
   cout << endl;
   fun<double>(1.1);
   cout << endl;
   return 0;
}
```

Practice question

x = 1 count = 0

x = 1 count = 1

x = 1.1 count = 0

Compiler creates a new instance of a template function for every data type. So compiler creates two functions in the above example, one for int and other for double. Every instance has its own copy of static variable. The int instance of function is called twice, so count is incremented for the second call.

Practice question

Which of the following is correct about templates in C++?
(1)When we write overloaded function we must code the function for each usage.
(2)When we write function template we code the function only once.

1. 1 only
2. 2 only
3. Both are true
4. Both are false

Practice question

Which of the following is correct about templates in C++?
(1)When we write overloaded function we must code the function for each usage.
(2)When we write function template we code the function only once.

1. 1 only
2. 2 only
3. Both are true
4. Both are false

Practice question

What will be the output of the following ?

```
#include <iostream>
using namespace std;

template <typename T>
T max(T x, T y)
{
    return (x > y)? x : y;
}
int main()
{
    cout << max(3, 7) << std::endl;
    cout << max(3.0, 7.0) << std::endl;
    cout << max(3, 7.0) << std::endl;
    return 0;
}
```

Output:

1. Compiler error in all cout statements
2. Compiler error in last cout statements
3. 7
   7.0
   7.0
4. Runtime error

Practice question

What will be the output of the following ?

```
#include <iostream>
using namespace std;

template <typename T>
T max(T x, T y)
{
    return (x > y)? x : y;
}
int main()
{
    cout << max(3, 7) << std::endl;
    cout << max(3.0, 7.0) << std::endl;
    cout << max(3, 7.0) << std::endl;
    return 0;
}
```

Output:

1. Compiler error in all cout statements
2. Compiler error in last cout statements
3. 7
   7.0
   7.0
4. Runtime error

QNA Time

Any Questions ??

Any Questions??

Thank You!

See you guys in next class.