Name - Aniket Kumar
Registration no. - 12108348
Roll no. - RD2112B81

QNo. - 1

Q1. How to implement the concept of inheritance in template? Explain with example.

Ans. We can implement the concept of inheritance in template also as we do normally.
We can achieve these concept of inheritance in this way -

A. template to template
B. class to template
C. Template to class

Implementation of template to template

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

template <class T> //creating a template class
class Base{      //Base class
    Public:
        void Print(T a){ //member function.
            Cout<< a <<endl;
        }
};
template <class T> // creating another template class
class B: Public Base<T>{ //derived template class,
};                       //Inheriting a Base template
                         // class.

int main(){
    B <void> Obj; //creating object with void return type.
    Obj.Print(5); // calling the function by giving Parameter
}
```

Output: 5

Implementation of class to template

```
#include <iostream>
using namespace std;
class Base{     //creating normal base class
};
template <class T>      //creating template class
class Derived: Public Base{  //inheriting template class
    Public:                  //from normal class
        void Print(T a){ //member function having template
            cout<<a;     //variable.
        }
};
int main(){
    Derived <int> obj;
    obj.Print(5);
}
```

Output: 5

Implementation of template to class

```
#include <iostream>
using namespace std;
template <class T>  //creating a template class
class Base{         //Base template class
    Public:
        void Print(T a){ //member function having one
            cout<<a<<endl; //Parameter.
        }
};
```

~~Output~~ class B: Public Base<int>{
    // Normal class inheriting a template class with a return
    // type - int
};
int main(){
    B ob; // Creating object of derived class
    ob.Print(5); // Calling member function with int
                 // Parameter.
}

Output:- 5

In this way, we can implement the concept of inheritance in template.

—

Q2. write a Program to achieve the following:
a. implementing overloading of function template
b. implementing re-throwing exceptions.

⇒ Implementing overloading of function template.

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

template <class T> //making template with Placeholder T
T Mul(T a, T b){
    //Template function to multiply 2 numbers
    return a+b;
};

template<class T> //creating another template with Placeholder
T Mul(T a,T b,T c){ //T
    //Template function to multiply 3 numbers.
    //Previous function is overloaded here by increasing the
    //no. of Parameters.
    return a+b+c;
};

int main(){
    cout<<Mul(7,8)<<endl; //calling functions with different
    cout<<Mul(7,8,9)<<endl; //number of Parameters.
    return 0;
}
```

Output:-
56
504

Implementing re-throwing exceptions

```
#include <iostream>
using namespace std;
class Rethrowing{
    public:
        void display(){
            //member function of the class
            try{  // outer try block
                try{  // inner try block
                    throw 1;  // throwing the exception
                }
                catch(...){  // inner catch block
                    throw 'a';  // re-throwing the exception.
                    cout<<"Inner Catch";
                }
            }
            catch(...){  //outer catch block
                cout<<"outer Catch";  //Catching the re-
                                      //thrown exception
            }
        }
};
int main(){
    Rethrowing r;  // creating the object
    r.display();  // Calling the function
}
```

Output: outer Catch

Here, in case of re-throwing the exception, it is handled by outer catch block.

Q3 Differentiate between function template and macros by using examples.

=>

Function Template	Macros
• It is used to create a common function for different datatypes.	• It is a symbolic constant used for repetition kind of thing.
• It is a kind of function created by template keyword.	• It is created by #define Preprocessor statement.
• It is handled by compilers	• It is handled by Pre-Processor.
• It takes longer execution time.	• It is faster to execute than that of template.

Example of function template

```
#include<iostream>
using namespace std;
template <class T> // creating template
T display(T a){ // template function which can get any
    return a;   // data type variable and return the same.
}
int main(){
    cout<<display(5)<<endl;
    // Calling function with Parameters of different data type.
    cout<<display("Aniket")<<endl;
    cout<<display(29.56)<<endl;
    return 0;
}
```

Output:
5
Aniket
29.56

Output is displayed for different data type using same function.

Example of Macros

```
#include <iostream>
using namespace std;
#define PI 3.14 // Here we are defining 'PI' symbolic
                // constant which replaces the value
                // of 3.14
int main(){
    cout<<"Area of Circle with radius 5 is "<<PI*r*r;
    return 0;
}
// Here, we are using the PI macro while calculating the
// area.
```

Output:
Area of Circle with radius 5 is 78.5.

Thus, function template and macros are slightly similar to each other but they are quite different in terms of use and execution.