Unit-7 Function Templates and Exception Handling

Generic – Template function

- Are features of C++ programming language that allow functions and class to operate with generic type.
- Two kinds of templates:
    1. Function template: templates declared for function.
    2. Class template: templates declared for classes.

- it allows a single template to deal with a generic data type.

1. Function Template:

Syntax:

```
Template < class T, ....>
Returntype func_name(args)
{
        //body of template func;
}
```

Eg: function template

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

template <class T>
T GetMax (T a, T b) {
  T result;
  result = (a>b)? a : b;
  return (result);
}

int main () {
  int i=5, j=6, k;
  long l=10, m=5, n;
  k=GetMax<int>(i,j);
  n=GetMax<long>(l,m);
  cout << k << endl;
  cout << n << endl;
  return 0;
}
```

```
6
10
```

Function Template with multiple parameters:

Syntax:

```
template<class T1, class T2,.....>

return_type function_name (arguments of type T1, T2....)

{

   // body of function.

}
```

```
1  #include <iostream>
2  using namespace std;
3  template<class X,class Y> void fun(X a,Y b)
4  {
5      cout << "Value of a is : " <<a<< endl;
6      cout << "Value of b is : " <<b<< endl;
7  }
8
9  int main()
10 {
11    fun(15,12.3);
12
13    return 0;
14 }
```

```
Value of a is : 15
Value of b is : 12.3
```

Overloading a Function Template:

```
1  //overloading a function template
2  #include <iostream>
3  using namespace std;
4  template<class X> void fun(X a)
5  {
6      cout << "Value of a is : " <<a<< endl;
7  }
8  template<class X,class Y> void fun(X b ,Y c)
9  {
10     cout << "Value of b is : " <<b<< endl;
11     cout << "Value of c is : " <<c<< endl;
12 }
13 int main()
14 {
15    fun(10);
16    fun(20,30.5);
17    return 0;
18 }
```

```
Value of a is : 10
Value of b is : 20
Value of c is : 30.5
```

2. Class Template:

- Class can also be declared to operate on different data types such class are called class template.
- Syntax:

```
Template <class T1, class T2,…>
Class class_name
{
        attributes;
        methods;
};
```

```
Template <class T> // T=any data type
Class add
{
        public:
        T add (T,T);
};
```

```cpp
#include <iostream>
using namespace std;
template <class T>
class Calculator {
   private:
    T num1, num2;
public:
    Calculator(T n1, T n2) {
        num1 = n1;
        num2 = n2; }
void displayResult() {
        cout << "Numbers: " << num1 << " and " << num2 << "." << endl;
        cout << num1 << " + " << num2 << " = " << add() << endl;
        cout << num1 << " - " << num2 << " = " << subtract() << endl;
        cout << num1 << " * " << num2 << " = " << multiply() << endl;
        cout << num1 << " / " << num2 << " = " << divide() << endl; }
 T add() { return num1 + num2; }
    T subtract() { return num1 - num2; }
    T multiply() { return num1 * num2; }
    T divide() { return num1 / num2; }};
int main() {
    Calculator<int> intCalc(2, 1);
    Calculator<float> floatCalc(2.4, 1.2);
cout << "Int results:" << endl;
    intCalc.displayResult();
cout << endl
        << "Float results:" << endl;
    floatCalc.displayResult();
 return 0;}
```

input

```
2 * 1 = 2
2 / 1 = 2

Float results:
Numbers: 2.4 and 1.2.
2.4 + 1.2 = 3.6
2.4 - 1.2 = 1.2
2.4 * 1.2 = 2.88
2.4 / 1.2 = 2
```

Template and Inheritance

```cpp
//templateInheritance2.cpp

#include <iostream>
using namespace std;
template <typename T>
class Base{
public:
  void func1() const {
    cout << "func1()\n";
  }
  void func2() const {
    cout << "func2()\n";
  }
  void func3() const {
    cout << "func3()\n";
  }
};
template <typename T>
class Derived: public Base<T>{
public:
  using Base<T>::func2;                // (2)
  void callAllBaseFunctions(){

    this->func1();                     // (1)
    func2();                           // (2)
    Base<T>::func3();                  // (3)

  }
};
```

```
func1()
func2()
func3()
```

Exceptional Handling in C++

- Exception means an error.
- Is the process of converting system error messages into user-friendly error messages is known as exception handling.
- It is an event, which occurs during the execution of the program, that disrupts the normal flow of the program instruction.
- Error can be classified into two types:
  - **Compile time error:** error caught during compile time. It includes library references, syntax errors, or incorrect class input.
  - **Run time error:** also called as exception. An exception caught during run time creates serious issues.
- **Eg:** user divides a number of zero (x/0), this will compiles successfully but an exception or runtime error will occur due to which our application will crashed.

- Inorder to avoid this we will introduce exceptional handling techniques in our code.
- **Exception handling Mechanism:**
  - find the problem (hit the exception).
  - Inform about its occurrence (throw the exception)
  - Receive error information (catch the exception)
  - Take the proper action (handle the exception)
- **C++ error handling keywords:**

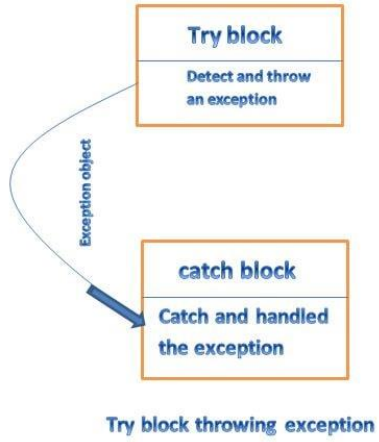
1. **Try**
2. **Catch**
3. **Throw**

Syntax:

```
try {
// Block of code to try
throw exception; // Throw an exception when a problem arise
}
catch () {
// Block of code to handle errors
}
```

Try, Catch and Throw

- Try block is intended to throw exception which is followed by catch block, only one try block.
- Catch block is intended to catch the error and handle the exception. We can have multiple catch blocks.
- Throw is a keyword that throws an exception encounter inside the try block. It is used to Communicate information about errors.



Try block throwing exception

Eg: here the program compiles successfully but the program fails during runtime.

```
#include <iostream>
Using namespace std;
Int main(){
Int a =50, b=0, c;
C= a/b; // error occurs during run time (50/0) ;the program will crash.
Return 0;
}
```

Implementation of try catch and throw statements:

```
Include <iostream>
Using namespace std;
Int main() {
Int a=50,b=0,c;
Try { // activates the exception handling.
If(b==0)
Throw “division by zero is not possible”;
C = a/b;
}
Catch(char *expression) // it catches the exception raised by the try block
{ cout <<expression;}
Return 0;
}
```

Output

0

Multiple catch statements:

```cpp
# include<iostream>
using namespace std;
int main(){
int n1,n2, result;
cout<<"enter 1st no.=";
cin>>n1;
cout<<"enter 2nd no.=";
cin>>n2;
try
{
if(n2==0)
throw n2; //throw statement 1
else
result = n1/n2;
cout<<"the result is the ="<<result;
}
catch (int x) {
cout<<"can not divide by:"<<x; }
cout<<"end of the program";
}
```

input

```
enter 1st no.=20
enter 2nd no.=40
the result is the =0end of the program
```

Multiple Catch blocks

```cpp
# include<iostream>
using namespace std;
int main()
{
    int a=2;
try
{
if(a==1)
throw a; //throwing int exception
else
if (a==2)
throw 'A'; // throwing char exception
else
if (a==3)
throw 2.4; //throwing float exception
}
catch(int a) {
cout<<"int exception caught";
    }
catch(char ch)
{
cout<<"char exception caught";
}
catch(double d)
{
cout<<"double exception caught";
}
cout<<"end of the program";
return 0;
}
```

```
char exception caughtend of the program
```

Catching all exceptions

```cpp
#include <iostream>
using namespace std;
int main()
{
    int x = 1;
    char a = 'a';
    try{
        if(x == 0){
            throw x;;
        }
        if(a == 'a'){
            throw a;
        }
    }
    catch(...){
        cout << "Exception Caught";
    }
    return 0;
}
```

```
Exception Caught
```

Why Exception Handling?

- No exception handling
  - Unchecked errors abort the program
  - Clutter program with if-clauses checking for errors
  - Use the return value to indicate errors
  - Pass an error-label parameter in all subprograms
  - Pass an errors handling subprogram to all subprograms
- With exception handling
  - Programs can catch exceptions, handle the problems and continue
  - Programmers are encouraged to consider all possible errors
  - Exception propagation allows for reuse of exception handling code

Multiple Choice Questions

1. Which of these built-in data types cannot be passed as non-type template parameter?
   (a) `int` (b) `char` (c) `long` (d) `double`
2. A generic class is also known as _____.
   (a) class template (b) template class (c) generated class (d) instantiated class
3. Which of these statements is true for a class template?
   (a) A class template is instantiated when its objects are created.
   (b) The public members of the class template are accessed using the object name and the dot operator.
   (c) The definition of a class template is prefixed by the template statement.
   (d) all of these
4. Which of these is the correct syntax for creating an instance of a class template?
   (a) `classtemplate_name_type object_name(argument_list);`
   (b) `classtemplate_name<type> object_name(argument_list);`
   (c) `classtemplate_name object_name(argument_list);`
   (b) none of these
5. The statement that informs the compiler that a template is being declared is known as _____
   (a) class template (b) function template (c) template statement (d) none of these

EXERCISES

Fill in the Blanks

1. The exceptions that are issued by external sources which are beyond the control of the program are called _____ .
2. The conventional error-handling method does not provide a way to separate the _____ code from the code written to handle the other tasks of the program.
3. C++ exception handling mechanism mainly uses three keywords, _____, _____ and _____ .
4. If an appropriate catch block is not executed, the standard library function _____ is invoked, which by default calls the _____ function to terminate the program abnormally.
5. The _____ statement without any argument is used to rethrow the exception caught by the catch block.

Multiple Choice Questions

1. An exception is caused by a _____.
   (a) syntactical error (b) logical error (c) semantic error (d) run-time error
2. Which of these is not a correct form of the throw statement?
   (a) `throw<exception>` (b) `throw (exception)` (c) `throw exception` (d) `throw`
3. If an appropriate catch block is not found, the current function is popped from the stack and the control passes to the next outer try/catch block. This process is called _____
   (a) stack popped (b) pop stack (c) stack unwinding (d) all of these
4. An exception cannot be thrown from
   (a) a try block in a function
   (b) a return statement of the function
   (c) a throw statement in a catch block
   (d) a function called in a try block
5. Which of these errors is not an exception?
   (a) the hardware interrupts such as pressing `Ctrl-C`
   (b) invalid array index
   (c) stack overflow
   (d) a power failure

1 d 2. a 3 d 4. b 5 c

1 Asynchronous exceptions 2 error handling 3 try, catch, throw

4 terminate(), abort() 5 throw

1 d 2 a 3 c 4 b 5 d