Unit-5 Templates and Exception Handling

5.1 Exception Handling

5.1.1 Need of an Exception

- Usually there are mainly two types of bugs, logical errors and syntactic errors.
- The logical errors occur due to poor understanding of problem and syntactic errors arise due to poor understanding of language.
- There are some other problems called exceptions that are run time anomalies or unused conditions that a program may encounter while executing.
- These anomalies can be division by zero, access to an array outside of its bounds or running out of memory or disk space.
- When a program encounters an exceptional condition it is important to identify it and dealt with it effectively.
- An exception is an object that is sent from the part of the program where an error occurs to that part of program which is going to control the error.
- The purpose of exception handling mechanism is to detect and report exceptional circumstances so that appropriate action can be taken.

5.1.2 Exception Handling Overview

- Exceptions are basically of two types namely, synchronous and asynchronous exceptions.
- Errors such as “out of range index” and “over flow” belong to synchronous type exceptions.
- The errors that are caused by the events beyond the control of program(such as keyboard interrupts) are called asynchronous exceptions.
- The purpose of exception handling mechanism is to detect and report an exceptional circumstances so that appropriate action can be taken.
- The mechanism for exception handling is
    - 1. Find the problem (hit the exception).
    - 2. Inform that an error has occurred (throw the exception).
    - 3. Receive the error information (Catch the exception).

- 4. Take corrective actions (Handle the exception).

- The error handling code mainly consist of two segments,one to detect error and throw exceptions and other to catch the exceptions and to take appropriate actions.

5.1.3 Exception Handling Mechanism

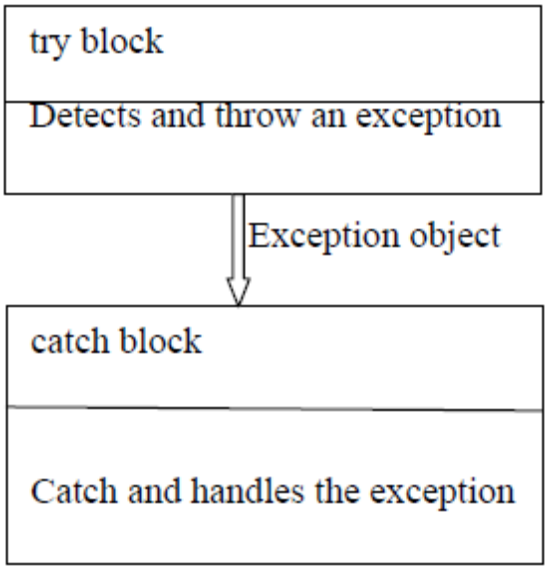
- C++ exception handling mechanism is basically built upon three keywords namely try, throw and catch.
- The keyword try is used to preface a block of statements which may generate exceptions.
- This block of statement is called try block. When an exception is detected it is thrown using throw statement in the try block.
- A catch block defined by the keyword catch who 'catches' the exception thrown by the throw statement in the try block and handles it appropriately.
- The catch block that catches an exception must immediately follow the try block that throws the exception.

Syntax: -

```
…….
……..
try
{
……
…….. //block of statements which detects and throw an exceptions
throw exception;                    //throw Exception to catch block
…….
…….
}
catch(type arg) //catches exceptions
{
…… // Block of statements that handles the exceptions
…….
…….
}
……
```

……..

- When the try block throws an exception, the program control leaves the try block and enters the catch statement of the catch block.
- If the type of object thrown matches the arg type in the catch statement, then the catch block is executed for handling the exception.
- If they donot match, the program is aborted with the help of abort() function which is executed implicitly by the compiler.
- When no exception is detected and thrown, the control goes to the statement immediately after the catch block i.e catch block is skipped.
- The below diagram 9.1 will show the mechanism of exception handling



- **Example:**

```
#include<iostream>
using namespace std;
int main()
{
int a,b;
cout<<"enter the values of a and b";
cin>>a;
cin>>b;
try
```

```
{
if(b!=0)
{
cout<<"result(a/b)="<<a/b<<"\n";
}
else
{
throw(b);
}
}
catch(int i)
{
cout<<"exception caught : b = "<<i<<"\n";
}
cout<<"end";
return 0;
}
```

The program detects and catches a division by zero problem.

5.1.4 Exception inside Function

- The exceptions are thrown by functions that are invoked from within the try block.
- The point at which the throw is executed is called throw point.
- Once an exception is thrown to catch block, control cannot return to the throw point.
- The general format of code for this kind of relationship is shown below

```
type function (arg list) //function with exception
{ ……..
……
throw(object); //throw exception
……..
……..
}
……
```

…….

try

{ …….

……. Invoke function here

……..

}

catch(type arg) //catches exception

{

……..

……. Handle exception here

…….

}

……..

……..

- It is to be noted here that the try block is immediately followed by the catch block irrespective of the location of the throw point.
- The below program demonstrates how a try block invokes a function that generates an exception

Example:

```
#include<iostream>
using namespace std;
void divide(int x,int y)
{
if(y!=0)
{
        cout<<"result(a/b)="<<x/y<<"\n";
}
else
{
        throw(y);
}
```

```
}

int main()
{
int a,b;
cout<<"enter the values of a and b";
cin>>a;
cin>>b;
try
{
        divide(a,b);
}
catch(int i)
{
cout<<"exception caught : b = "<<i<<"\n";
}
cout<<"end";
return 0;
}
```

5.1.4 Throwing Mechanism

- When an exception is encountered it is thrown using the throw statement in the following form:
  throw (exception);
  throw exception;
  throw;
- The operand object exception may be of any type including constants.
- It is also possible to throw objects not intended for error handling.
- When an exception is thrown, it will be caught by the catch statement associated with the try block.
- In other words the control exits the try block and transferred to catch block after the try block.

5.1.5 Catching Mechanism

- Code for handling exceptions is included in catch blocks. The catch block is like a function definition and is of form

  Catch(type arg)

  {

  statements for managing exceptions;

  }
- The type indicates the type of exception that catch block handles. The parameter arg is an optional parameter name.
- The catch statement catches an exception whose type matches with the type of catch argument.
- When it is caught, the code in the catch block is executed.
- After executing the handler, the control goes to the statement immediately following in catch block.
- Due to mismatch, if an exception is not caught abnormal program termination will occur. In other words catch block is simply skipped if the catch statement does not catch an exception.

5.1.6 Multiple Catch Statements

- In some situations the program segment has more than one condition to throw an exception.
- In such case more than one catch blocks can be associated with a try block as shown below:

```
try
{
        //try block
}
catch(type1 arg)
{
        //catch block1
}
```

```
catch(type 2 arg)
{
        //catch block 2
}
……..
…….
catch (type N arg)
{
        //catch block N
}
```

- When an exception is thrown, the exception handlers are searched in order for an appropriate match. The first handler that yields a match is executed.
- After executing the handler, the control goes to the first statement after the last catch block for that try.
- When no match is found, the program is terminated.
- If in some case the arguments of several catch statements match the type of an exception, then the first handler that matches the exception type is executed.

Example:

5.2 Templates

- Template is a new concept which enables us to define generic and functions and thus provides support for generic programming.
- Generic programming as an approach where generic types are used as parameters in algorithms so that they work for a variety of suitable data types and data structures.
- A template can be used to create a family of classes or functions.
- For example, a class template for an array class would enable us to create arrays of various data types such as int array and float array.
- Similarly, we can define a template for a function, say mul(),that would help us create versions of mul() for multiplying int, float and double type values.
- A template can be considered as a kind of macro. When an object of a specific type is define for actual use, the template definition for that class is substitute with the required data type.
- Since a template is defined with a parameter that would be replaced by a specified data type at the time of actual use of the class or function, the templates are sometimes called parameterized class or functions.

5.2.1 Function Template

- We write a generic function that can be used for different data types. Examples of function templates are sort(), max(), min(), printArray().

Example:

```
#include <iostream>
using namespace std;
// One function works for all data types. This would work
template <typename T>
T myMax(T x, T y)
{
return (x > y)? x: y;
}
int main()
{
```

```
cout << myMax<int>(3, 7) << endl; // Call myMax for int
cout << myMax<double>(3.0, 7.0) << endl; // call myMax for double
cout << myMax<char>('g', 'e') << endl; // call myMax for char
return 0;
}
```

5.2.2 Class Template

Class Templates Like function templates, class templates are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, Array, etc.

Example:

```
#include <iostream>
using namespace std;
template <typename T>
class Array
{
private:
    T *ptr;
    int size;
public:
    Array(T arr[], int s)
    {
        ptr = new T[s];
        size = s;
        for(int i = 0; i < size; i++)
            ptr[i] = arr[i];

    }
    void print()
    {
        for (int i = 0; i < size; i++)
            cout<<" "<<*(ptr + i);
```

```
                cout<<endl;
        }
};

int main()
{
        int arr[5] = {1, 2, 3, 4, 5};
        Array<int> a(arr, 5);
        a.print();
        return 0;
}
```