Unit

11

Exception Handling

1. Introduction

Errors are an intrinsic part of programming. An error typically refers to a problem that exists in a program. It could be a syntax error or a logical error. A syntax error occurs when the program does not follow the syntax of the language. The compiler finds errors of this type and they can be corrected easily. A logical error occurs due to an incorrect algorithm. In this case, the program runs but often gives an incorrect output. This type of error will only be found during program testing and debugging.

However, we often come across some peculiar problems other than logical or syntax error. They are known as exceptions. They occur during runtime. When an exception occurs, the program stops. Hence, there should be some mechanism to monitor the program for such situations so that appropriate action can be taken. In this chapter we will focus on the concept of an Exception and exception handling mechanism in C++ using keywords try, catch and throw. Exception handling is a new feature added to ANSI C++ and it was not part of the original C++.

2. What is an Exception?

Exceptions are run-time errors or anomalies that occur during the execution of a program. An *exception* is an event, which occurs during the execution of a program and disrupts the normal flow of the program's instructions. It causes the program to terminate abnormally.

It is caused due to the unavailability of system resources, such as memory, file space or from data dependent conditions such as division by zero, or numerical overflow or illegal access to a memory location. Such exceptions require immediate handling by the program.

3. Exception Handling Mechanism

Exception handling is a mechanism to detect and report exceptions so that appropriate action can be taken. The exception handling is done in the following steps:

1. Hit the exception. In other words, find the problem.
2. Throw the exception i.e. report the exception.
3. Catch the exception. It means, receive information about the exception.
4. Handle the exception i.e. take corrective measures against the problem.

The mechanism is as follows:

When a function detects an exceptional situation, the situation is represented by an object. This object is called an *exception object*. In order to handle the exception, you *throw the exception*. This passes control, to a designated block of code which is called a *handler*. Control is passed to the appropriate handler by the C++ run time system. When this happens, the exception is *caught*. A handler may *rethrow* an exception so it can be caught by another handler.

If no exception arises, the program executes normally.

In C++, the exception handling is done using three keywords:

1. try
2. throw
3. catch

The try-throw-catch sequence

The general form of the try and catch block are shown below

```
try {
    ...
    ...
    throw Exception;   //Detect and throw exception
    ...
    ...
}
catch(type arg)     // Catches exception
{
    ...                 // Code that handles the exception
    ...
}
```

The try Block

A try block is a group of C++ statements, enclosed in braces { }, which may cause an exception. The general form of the try block is as follows:

```
try
{
  // statements in which exceptions may arise
}
```

A try block can contain any C++ statement such as expressions as well as declarations. A try block has local scope, and variables declared within a try block cannot be referred to outside the try block.

The throw Statement

The throw statement is used to throw an exception which occurs inside a try block to an exception handler. When an exception is detected inside a try block, it is thrown using the *throw* keyword.

The throw statement takes one of the following forms:

```
throw(exception);
throw exception;
throw;       // used to rethrow an exception
```

where *exception* is the object which contains information about the problem. The first two forms are used to throw a value or an expression of a specific type. The third form is used to rethrow the exception.

For example,

throw expression	Effect
throw 4;	The constant value 4 is thrown.
throw num;	The value of the variable num is thrown.
throw str;	The object str is thrown.
throw ErrorMessage("Exception found!");	An object of user defined class ErrorMessage with the string "Exception found!" is thrown.
throw;	Rethrow the exception.

The type of the exception which is thrown determines which catch block to execute.

In some cases, the catch block which catches an exception may not be the correct place to take necessary action. If a catch block cannot handle the particular exception, it can rethrow the exception. The rethrow expression causes the original exception object to be rethrown to the enclosing try-catch block.

The rethrow expression has following form

```
throw;
```

This causes the current exception to be passed to the outer try/catch sequence in hierarchy. If there is no outer-try catch block, the exception is rethrown to the calling function.

▶The catch block: Exception Handler

A catch block is a group of C++ statements that are used to handle a specific exception. Catch blocks, or handlers, should be placed immediately after the try block.

A catch block is specified by:

1. The keyword catch.
2. A catch expression, which corresponds to a specific type of exception that may be thrown by the try block.
3. A group of statements, enclosed in braces { }, whose purpose is to handle the exception.

Syntax of catch block

```
catch(type arg)
{
   //code that handles the exception
}
```

For example: A catch-block receiving char * exceptions

```
catch(int value)
{
   // code in which int type exceptions are handled
}
```

The following program is a simple *example* of a **try - catch**. In this program, we accept two integers a and b from the user and divide a by b. If the user enters a positive value for b, the result will be displayed. If the user enters 0 for b, we will throw an exception and pass control to the exception handler (by skipping the remaining statements in the try block) and display an appropriate message on the screen.

Program : To illustrate try-catch

```
#include<iostream.h>
int main()
{
  int a,b;
  try
  {
     cout<<"In try block"<<endl;
     cout<<"Enter the values of a and b : ";
     cin >> a >> b;
     if(b != 0)
        cout << a << "/" << b <<"=" << a/b <<endl;
     else
      throw (b);
     cout<<"End of try block"<<endl;
  }
  catch(int n)
  { cout << "Exception caught: Value = " << n << endl;
  }
  cout<<"End of main"<<endl;
  return 0;
}
```

Output 1

```
In try block
Enter the values of a and b:
20  5
20/5=4
End of try block
End of main
```

Output 2

```
In try block
Enter the values of a and
b: 10  0
Exception caught:Value=0
End of main
```

In the above program, the block which causes the exception and the block which handles the exception are placed in the same function (in this case, main). However, the statements which can cause an exception could be in a different function too. The following program shows how this will work.

Program : Function Throwing an exception

```
#include<iostream.h>
int division(int a, int b)
{
   cout<<"Inside function"<<endl;
      if(b != 0)
        cout << a << " /" << b <<"=" << a/b <<endl;
```

```
    else
      throw(b);
}
int main ()
{
  int a,b;
  try
  {
     cout<<"In try block"<<endl;
     cout<<"Enter the values of a and b : ";
     cin >> a >> b;
     division(a,b);
     cout<<"End of try block"<<endl;
 }
  catch(int n)
  {
    cout << "Exception caught: Value = " << n << endl;
  }
  cout<<"End of main"<<endl;
  return 0;
}
```

Output 1

In try block
Enter the values of a and b: 20 5
Inside function
20/5=4
End of try block
End of main

Output 2

In try block
Enter the values of a and b: 20 0
Inside function
Exception caught: Value = 0

4. Multiple Catch Statements

A try block can be followed by one or more than one catch blocks. Each catch block must handle a different type of exception. Only the matching catch block will be executed. If no match is found, the program terminates abnormally.

The general form of multiple catch statement is as follows

```
try
{
     //code which throws an exception
}
catch(type1 arg)
{
     //catch block1
}
catch(type2 arg)
{
     //catch block2
}
. . .
catch(typeN arg)
{
     //catch blockN
}
```

When an exception is thrown, the catch blocks are searched sequentially for an appropriate match – similar to a switch case statement. The first handler that matches is executed. After executing the handler, the remaining catch blocks are bypassed and the control goes to the first statement after the last catch block. If no match is found, the exception is thrown to the outer try-catch if any or to the calling function. If it is not handled there, the program terminates. If arguments of several catch statements match the type of an exception thrown by the try block, the first handler that matches the exception type is executed.

Program : Multiple catch blocks

```
#include<iostream.h>
int main()
{
   int i;
   cout<<"Multiple Catch Statements\n";
   try
   {
      cout<<"Enter an integer 1/2/3/4: ";
      cin >> i;
         if(i==1)    throw 10;      //int
      else
         if(i==2)    throw '*';     //char
      else
         if(i==3)    throw 2.15;    //double
      else
         if(i==4)
            cout<<"Normal execution"<<endl;
   }
   catch(char c)       //catch1
   {
     cout<<"Caught a character"<<c<<endl;
   }
   catch(int x)          //catch2
   {
       cout << "Caught an integer "<<x<<endl;
   }
   catch(double f)     //catch3
   {
      cout << "Caught a double "<<f <<endl;
   }
   cout<<"End of try-catch \n";
   return 0;
}
```

Output 1

```
Multiple Catch Statements
Enter an integer 1/2/3/4: 1
Caught an integer 10
End of try-catch
```

Output 2

```
Multiple Catch Statements
Enter an integer 1/2/3/4: 2
Caught a character *
End of try-catch
```

Output 3

```
Multiple Catch Statements
Enter an integer 1/2/3/4: 3
Caught a double 2.15
End of try-catch
```

Output 4

```
Multiple Catch Statements
Enter an integer 1/2/3/4: 4
Normal Execution
End of try-catch
```

▶ Generic Catch Block

It may not always be possible to identify all possible types of exceptions that a program might throw. Hence, it may not be possible to define independent handlers for each type. In such cases, we can define a single catch which can catch exceptions of all types. If we use an ellipsis (...) as the parameter of catch, that handler will catch any exception no matter what is the type of the exception.

Its form is as follows:

```
catch (. . .)
{
   //process all exceptions
}
```

Here, the ellipsis matches any type of data. However, there are some points that need to be noted regarding this:

1. If catch(...) is the only exception handler, it can catch all exception types.
2. The catch(...) can accompany other catch blocks.
3. It should be placed as the last catch block otherwise no other catch block will be executed.

Program : Catch all exception types

```
#include<iostream.h>
int main()
{
  cout << "Begin\n";
  cout << "Enter a number 1/2/3 :";
  cin >> i;
  try
  {
    if(i==1) throw i;      // throw int
    if(i==2) throw '*';    // throw char
    if(i==3) throw 2.12;   // throw double
  }
  catch(...)  //catch all exceptions
  {
    cout << "Caught an exception\n";
  }
  cout << "End";
  return 0;
}
```

Output 1

Begin
Enter a number 1/2/3: 3
Caught an exception
End

Output 2

Begin
Enter a number 1/2/3: 1
Caught an exception
End

5. Nested Try-catch Blocks

It is also possible to define try-catch blocks within other try blocks. If an exception is thrown by an outer try block, it is passed only to the outer catch blocks. However, if an exception is thrown by an inner try block, it is first thrown to the inner catch blocks. If it is handled there, the control passes to the statements after the outer catch blocks. If it is not handled by the inner catch block, it will be passed to the outer catch blocks.

```
try
{
   //outer try code here
   try
   {
       // inner try code here
   }
   catch(type arg)
   {
       // inner catch block
   }
}
```

```
catch(type arg)
{
  //outer catch block
}
```

The following program illustrates nested try-catch blocks.

Program: nested try-catch blocks

```
#include<iostream.h>
int main()
{
  int i;
  cout << "Enter a value 1/2 : ";
  cin>>i;
  try
  {
  try
  {  if(i==1)
       throw  "Welcome";       // throw a string
    else
       throw i;
  }
  catch(char *s)
  {  cout << " Inner catch, string = " << s <<endl;
    throw;       // re-throw the same exception
  }
}
catch(int x)
{
  cout << "Outer catch, integer = " << x <<endl;

atch(char * str)

 cout << "Outer catch, string = " <<str <<endl;

ut << "End";
turn 0;
```

Output 1

Enter a value 1/2: 1
Inner catch, string = Welcome
Outer catch, string = Welcome
End

Output 2

Enter a value 1/2: 2
Outer catch, integer = 2
End

EXERCISES

A. True or False

1. Once an exception is handled, control does not return to the try block.
2. A catch(...) should always be placed first in the list of handlers following a try block.
3. Exceptions can be thrown from anywhere in a program.
4. A try block should be followed by one or more catch blocks.

B. Review Questions

1. What is an Exception?
2. How is an exception handled in C++?
3. What is a try block?
4. What is a catch statement?
5. What are the different types of errors that can occur in a program?
6. What does catch(...) mean?
7. When do we use multiple catch handlers?
8. How is an exception rethrown?

C. Programming Exercises

1. Write a program containing a possible exception. Use a try block to throw an exception and catch block to handle it properly.
2. Write a program, which illustrates the use of multiple catch statements.
3. Write a program using catch(. . .) handler.
4. Write a program which accepts student information (name, age and year). The program should throw an exception for the following situations: age is not between 18 and 25, year is not "FY", "SY" or "TY".

Answers

A.

1. True
2. False
3. False
4. True

Notes