EXPERIMENT-12

OBJECTIVE

Write programs to understand exception handling techniques.

PROGRAM

Exception Handling: An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Errors can be broadly categorized into two types. We will discuss them one by one.

1. Compile Time Errors
2. Run Time Errors

Compile Time Errors – Errors caught during compiled time is called Compile time errors. Compile time errors include library reference, syntax error or incorrect class import.

Run Time Errors - They are also known as exceptions. An exception caught during run time creates serious issues.

Errors hinder normal execution of program. Exception handling is the process of handling errors and exceptions in such a way that they do not hinder normal execution of the system. For example, User divides a number by zero, this will compile successfully but an exception or run time error will occur due to which our applications will be crashed. In order to avoid this we'll introduce exception handling technics in our code.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try, catch,** and **throw**.

- **throw** − A program throws an exception when a problem shows up. This is done using a **throw** keyword.
- **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The **catch** keyword indicates the catching of an exception.
- **try** − A **try** block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Syntax:

```
try
{
   //code
   throw parameter;
}
catch(exceptionname ex)
{
   //code to handle exception
}
```

try block

The code which can throw any exception is kept inside(or enclosed in) a`try` block. Then, when the code will lead to any error, that error/exception will get caught inside the `catch` block.

catch block

`catch` block is intended to catch the error and handle the exception condition. We can have multiple catch blocks to handle different types of exception and perform different actions when the exceptions occur. For example, we can display descriptive messages to explain why any particular excpetion occured.

throw statement

It is used to throw exceptions to exception handler i.e. it is used to communicate information about error. A `throw` expression accepts one parameter and that parameter is passed to handler.

`throw` statement is used when we explicitly want an exception to occur, then we can use `throw` statement to throw or generate that exception.

Understanding Need of Exception Handling

Let's take a simple example to understand the usage of try, catch and throw.

Below program compiles successfully but the program fails at runtime, leading to an exception.

```
#include <iostream>#include<conio.h>

using namespace std;

int main()

{

   int a=10,b=0,c;

   c=a/b;

   return 0;

}
```

The above program will not run, and will show **runtime error** on screen, because we are trying to divide a number with **0**, which is not possible.

How to handle this situation? We can handle such situations using exception handling and can inform the user that you cannot divide a number by zero, by displaying a message.

Using try, catch and throw Statement

Now we will update the above program and include exception handling in it.

```
#include <iostream>

#include<conio.h>

using namespace std;

int main()

{
```

```
    int a=10, b=0, c;
    // try block activates exception handling
    try
    {
        if(b == 0)
        {
            // throw custom exception
            throw "Division by zero not possible";
            c = a/b;
        }
    }
    catch(char* ex) // catches exception
    {
        cout<<ex;
    }
    return 0;
}
```

Output:

Division by zero not possible.

In the code above, we are checking the divisor, if it is zero, we are throwing an exception message, then the `catch` block catches that exception and prints the message.

Doing so, the user will never know that our program failed at runtime, he/she will only see the message "Division by zero not possible".

This is **gracefully handling** the exception condition which is why exception handling is used.