Exception Handling

An exception is a runtime error. An exception may result in loss of data or an abnormal 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.

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 for error handling:

```
try {
 // protected code
 } catch( ExceptionName e1 ) {
 // catch block
 } catch( ExceptionName e2 ) {
 // catch block
 } catch( ExceptionName eN ) {
 // catch block
}
```

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.

```
try {
 // code
   throw parameter;
 } catch( ExceptionName e1 ) {
 // catch block
}
```

catch {} block

• This block catches the error thrown by try block. This block contains method to customize error.

```
 catch
{
//defines method to control error;
}
```

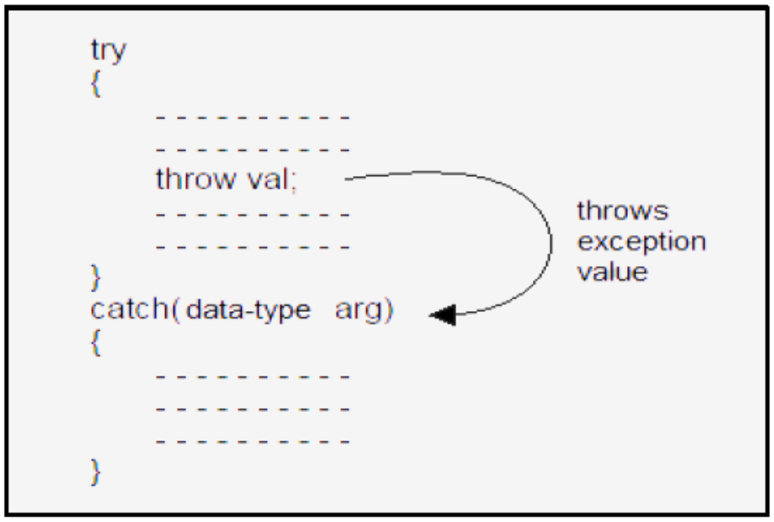
throw function

This function is used to transfer the error from try block to catch block. This function plays major role to save program from crashing.

Syntax:

```
throw(variable);
```

Exception flow



```
try
{
    ----------
    ----------
    throw val;
    ----------
    ----------
}
catch( data-type  arg)
{
    ----------
    ----------
    ----------
}
```

try-blocks and if-else

Try-blocks are very similar to if-else statements

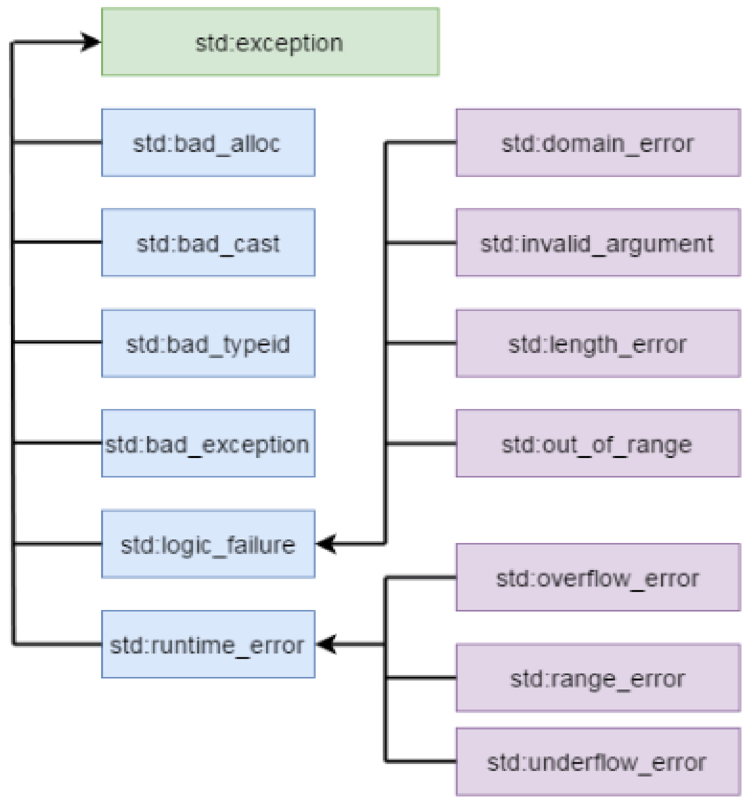
- If everything is normal, the entire try-block is executed
- else, if an exception is thrown, the catch-block is executed

Example Code:

```
int main()
```

```
{
int x[3] = {-1,2};
for(int i=0; i<2; i++){
int ex = x[i];
try
{
if (ex > 0)
// throwing numeric value as exception
throw ex;
else
// throwing a character as exception
throw 'ex';
}
catch (int ex) // to catch numeric exceptions
{
cout << "Integer exception\n";
}
catch (char ex) // to catch character/string exceptions
{
cout << "Character exception\n";
}
}
}
```

C++ Standard Exceptions



- std::exception – The exception and parent class of all the standard C++ exceptions.
- std::logic_error – This exception can be detected by reading the code.
- std::domain_error – This exception is thrown when an invalid domain is used.
- std::invalid_argument – This exception is thrown due to invalid arguments.
- std::out_of_range – This exception is thrown due to out of range which means size requirement exceeds the memory allocation.
- std::length_error – This exception is thrown due to length error.
- std::runtime_error – This exception occurs during runtime and cannot be detected by reading the code.
- std::range_error – This exception occurs when you try to store a value out of range.
- std::overflow_error – This exception occurs due to arithmetic overflow occurs.
- std::underflow_error – This exception occurs due to arithmetic underflow occurs.
- std::bad_alloc – This exception occurs when memory allocation fails by new( ).
- std::bad_cast – This exception occurs when dynamic cast fails.
- std::bad_exception – This exception is designed to be listed in the dynamic exception specifier.
- std::bad_typeid – This exception is thrown by typeid

User-defined Custom Exception with class in C++

Example Code:

```
class demo1 {
};
class demo2 {
};
int main()
{
for (int i = 1; i <= 2; i++) {
try {
if (i == 1)
throw demo1();
else if (i == 2)
throw demo2();
}
catch (demo1 d1) {
cout << "Caught exception of demo1 class \n";
}
catch (demo2 d2) {
cout << "Caught exception of demo2 class \n";
} } }
```

Exception handling with constructor

```
class demo {
int num;
public:
demo(int x) {
try {
if (x == 0)
throw "Zero not allowed "; // cout<<“Zero not allowed ”
num = x;
show();
}
catch (const char* exp) {
cout << "Exception caught \n ";
cout << exp << endl;}
}
void show()
{
cout << "Num = " << num << endl;
}
};
int main()
{
// constructor will be called
demo(0);
cout << "Again creating object \n";
demo(1);
}
```

Exception Handling in Filing

```
#include <fstream>
#include <iostream>
#include <vector>
#include <string>
#include <exception>
using namespace std;
void readIntegerFile(const string& fileName)
{
```

```
ifstream istr;
int temp;
char buffer[256];
istr.open(fileName.c_str());
if (istr.fail()) {
throw exception();
}
while (istr)
{
istr.getline (buffer,100);
cout << buffer << endl;
}
}
int main(int argc, char** argv)
{
const string fileName = "test66.txt";
try {
readIntegerFile(fileName);
} catch (const exception& e) {
cerr << "Unable to open file " << fileName << endl;
exit (1);
}
cout << endl;
return (0);
}
```

Practice Questions

Q1. Write a program that prompts the user to enter a length in feet and inches and outputs the equivalent length in centimeters. If the user enters a negative number or a non - digit number, throw and handle an exception and prompt the user to enter another set of numbers.

Q2. Write a program that prompts the user to enter time in 12 hour notation. The program then outputs the time in 24 hour notation. Your program must contain three exception classes: invalidHr, invalidMin and invalidSec. If the user enters an invalid value for hours, then the program should throw and catch an invalidHr object. Do the same for minutes and seconds.

Q3. Write a program that prompts the user to enter a person’s date of birth in numeric form such as 4-5-1987. The program then outputs the date in the format: May 4, 1987. Your program should contain atleast two exception classes: invalidDay and invalidMonth. If the user enters an invalid day value, the program should throw and catch an invalidDay object. Do the same for invalid values of month and year. Your program should also handle a leap year.