Exception Handling in C++

Exceptions

ISM
We inspire you to learn

Good program is stable and fault tolerant.

In a good program, **exceptional ("error") situations must be handled**.

- Exceptions are unusual events
- They are detectable by either hardware or software
- They may require special processing
- Exception handler is part of the code that processes an exception

Some examples for exceptions:

- EOF is reached while trying to read from a file
- Division by 0 is attempted (result is meaningless)
- Array subscript is out of range
- Bad input
- ...

Premier Embedded Training Centre in the country

Why exceptions?

In traditional code, the following may be done in error-prone situations:

- **error values are returned from a function**
  - **`new`** or **`malloc()`** returns null for out of memory
  - **`fopen()`** returns null when the file can not be opened

The programmer is then responsible for checking these returned values

Why exceptions?

ISM
We inspire you to learn

Example.

```
if ((p = malloc(sizeof(float)) == NULL)
        //handle error
if ( (f = fopen(”file.txt”, ”r”) ) = = NULL)
        //handle error
```

This kind of error detection

- makes the **program logic unclear**.
- **cannot be used to handle errors in constructors** (because they don’t return a value).
- user may not check the returned error
- errors must be handled immediately. The application should decide how to act in error situations.

Because of these drawbacks a new method to handle error situations (exceptions) is developed in C++.

Premier Embedded Training Centre in the country

Basics of Exception Handling

ISM
We inspire you to learn

A function can `throw` an exception object if it detects an error

- Object typically a character string (error message) or class object
- If exception handler exists, exception caught and handled
- Otherwise, program terminates

Premier Embedded Training Centre in the country

- Format
  - Enclose code that may have an error in `try` block
  - Follow with one or more `catch` blocks
    - Each `catch` block has an exception handler
  - If exception occurs and matches parameter in `catch` block, code in catch block executed
  - If no exception thrown, exception handlers skipped and control resumes after catch blocks
  - `throw` point - place where exception occurred
    - Control cannot return to `throw` point

Three Keywords

ISM
We inspire you to learn

- try: identifies a code block where exception can occur
- throw: causes an exception to be raised (thrown)
- catch: identifies a code block where the exception will be handled (caught)

Exception Handling: Sample Code

```
#include<iostream>
using namespace std;
float division(float i,float j)
{
cout<<"\n we entered into the division function...";
try
{
        cout<<"\n we are in try block...";
        if(j==0)
        throw j;
        return i/j;
}
catch(float)
{
        cout<<"\n we are in catch block...";
        cout<<"\n dividend value should not be 0.....";
}
cout<<"\n we are at the end of the division function..";
}
int main()
{
        float i,j,k;
        cout<<"\n we are in main block..";
        cout<<"\n enter two integers for division::";
        cin>>i>>j;
        k=division(i,j);
        cout<<"\n after division result is::"<<k;
        cout<<"\n";
        return 0;
}
```

Output:

```
we are in main block..
enter two integers for division::20   10
we entered into the division function...
we are in try block...
after division result is::2
we are in main block..
enter two integers for division::20   0
we entered into the division function...
we are in try block...
we are in catch block...
dividend value should not be 0.....
we are at the end of the division function..
after division result is::0
```


Premier Embedded Training Centre in the country

The fundamentals of exception handling

The ”abnormal” code is put into try block.

It means that we ”try to execute code” in the try block.

If the system succeeds to run the code, everything is fine.

If or when something goes wrong when code of try block is executed, this code throws an exception object and stops executing the code of try block further.

Another part of the code (the error handling part) can catch the exception (object) and make necessary actions needed in that error situation.

The exception object can contain information about the exception, so that the error handling part of the program can examine the reason and make appropriate actions.

Premier Embedded Training Centre in the country

If an exception is thrown

- **When an exception is thrown, the remaining code in the try block is skipped, just as in the case of the return statement in a function, and every object created within the try block is destroyed**
- **The thrown object is caught by the catch block where execution continues**
- **The execution continues with the next statement after the catch block**

Premier Embedded Training Centre in the country

Catching Exceptions

ISM
We inspire you to learn

- **You must supply at least one catch block for a try block**

```
try
{
    cout<<"\n we are in try block...";
    if(j==0)
    throw j;
    return i/j;
}
```

NOT ALLOWED!
**** we must have to write catch block immediately after try block.**

Premier Embedded Training Centre in the country

Catching Exceptions

ISM
We inspire you to learn

Catch blocks must immediately follow the try block without any program code between them.

```
try
{
    cout<<"\n we are in try block...";
    if(j==0)
    throw j;
    return i/j;
}
cout << “.....”;  // not allowed
catch(float)
{
    cout<<"\n we are in catch block...";
    cout<<"\n dividend value should not be 0.....";
}
```


Premier Embedded Training Centre in the country

Nested tries

ISM
We inspire you to learn

- In case of nested try blocks:
  - if an exception is thrown from within an inner try block which is not followed by a catch block with the right type, the catch handlers for the outer try block will be searched.

- If a catch block cannot handle the particular exception it has caught, it can rethrow the exception.

Premier Embedded Training Centre in the country

```
#include<iostream>
using namespace std;
void division(float i,float j)
{
        cout<<"\n=======================================\n";
        cout<<"\n we entered into the division function...";
        try
        {
                cout<<"\n we are in try block which is inside the function..";
                if(j==0)
                throw j;
                cout<<"\n after division result is::"<<i/j;
        }
        catch(float val)
        {
                cout<<"\n we are in catch block which is inside the function";
                cout<<"\n now we rethrowing the exception....";
                throw val;
        }
        cout<<"\n we are at the end of the division function..";
        cout<<"\n=======================================\n";
}
```

```
int main()
{
        cout<<"\n we are in main block..";
        try
        {
                cout<<"\n we are in try block which is inside main..";
                cout<<"\n we are passing the values 10.0 , 5.0";
                division(10.0,5.0);
                cout<<"\n we are passing the values 10.0 , 0.0";
                division(10.0,0.0);
        }
        catch(float)
        {
                cout<<"\n we are in catch block which is inside main..";
                cout<<"\n divisor must not be 0...";
        }

        cout<<"\n we are at the end of main";
        cout<<"\n";
        return 0;
}
```

Catching exceptions

- If you want to catch any exception that is thrown in a try block, you specify this as:

```
catch (…)
{
        // code to handle any exception
}
```

- This catch block must appear last if you have other catch blocks defined for the try block.
- Note that in this catch block, you do not know what type of exception has occured and cannot reference an object

Premier Embedded Training Centre in the country

Stack Unwinding

Stack unwinding

ISM
We inspire you to learn

If exception is thrown in a try block (in a function that is called from a try block or in a function that is called from a function that is called from a try block and so on), all local objects allocated from the stack after the try block was entered are released and their destructors are called. This process is called **stack unwinding**.

This process guaranties that when we try to recover from an error, there are no inconsistent data in the stack and try block can be started again if desired.

Premier Embedded Training Centre in the country

Stack unwinding

ISM
We inspire you to learn

Example 1.

```
void main() {
    ...
    try {
        A a;
        f1();
        ...
    }
    ...
}
void f1() {
    B b;
    f2();
    ...
}
void f2() {
    C c;
    ...
        throw Exception e;
}
```

If error occurs in the function f2, the destructors for objects a, b and c are called and those objects are deleted from the stack.

Premier Embedded Training Centre in the country

Exception Classes in C++

- C++ exception handling allows a clean separation between error detection and error handling.
  - A library developer may be able to detect when an error occurs, such as an argument out of range, but the developer doesn't know what to do about it.
  - You, the user, can't always detect an exceptional condition, but you know how your application needs to handle it.
- Hence, exceptions constitute a protocol for runtime error communication between components of an application.

Standard exception classes

ISM
We inspire you to learn

C++ standard defines the following class hierarchy for exceptions.

It is meant to serve as a starting point so that programmers can inherit and develop their own exceptions.

The base class for this class hierarchy is **exception**.

Premier Embedded Training Centre in the country

C++ Exception Classes

```
#include <iostream>
#include <exception>
using namespace std;
main()
{
  try {
     char *c = new char[10];
  }
  catch (std::exception & e)
  {
      cout << "Exception: " << e.what();
  }
}
```

Some functions of the standard C++ language library send exceptions that can be captured if we include them within a try block.

These exceptions are sent with a class derived from std::exception as type.

This class (std::exception) is defined in the C++ standard header file <exception> and serves as base for the standard hierarchy of exceptions

Standard exception classes

ISM
We inspire you to learn

Exception class hierarchy:

- **exception**
  - **bad_alloc**
  - **bad_cast**
  - **bad_typeid**
  - **logic_error**
    - domain_error
    - invalid_argument
    - length_error
    - out_of_range
  - **runtime_error**
    - range_error
    - overflow_error
    - underflow_error
  - **ios_base::failure**
  - **bad_exception**

A logic error indicates an inconsistency in the internal logic of a program, or a violation of pre-conditions on the part of client software.

For example, the substr member function of the standard string class throws an out_of_range exception if you ask for a substring beginning past the end of the string.

A bad_alloc exception occurs when heap memory is exhausted. C++ will generate a bad_cast exception when a dynamic_cast to a reference type fails.

If you rethrow an exception from within an unexpected handler, it gets converted into a bad_exception. If you attempt to apply the typeid operator to a null expression, you get a bad_typeid exception.

Premier Embedded Training Centre in the country

```
// exception example
#include <iostream>       // std::cerr
#include <typeinfo>       // operator typeid
#include <exception>      // std::exception

class Polymorphic {virtual void member(){}};

int main () {
  try
  {
    Polymorphic * pb = 0;
    typeid(*pb);  // throws a bad_typeid exception
  }
  catch (std::exception& e)
  {
    std::cerr << "exception caught: " << e.what() << '\n';
  }
  return 0;
}
```