C++ in depth

Exception handling

Saurabh Shukla (MySirG)

Agenda

1. Types of Errors
2. Exceptions
3. Why exception handling?
4. Exception Handling
5. try, throw, catch
6. multiple throw, multiple catch
7. Catch all
8. Defining own custom exception

Errors

1. Syntax Errors
2. Linker Errors
3. Runtime Errors
4. Logical Errors

Syntax Error

Syntax error is grammatical errors.

Linker Error

This type of error generates when there is an invalid library component reference used in the program or main() method is not present in the program.

Run time Errors

Error occur during execution of program.

For example, error due to an attempt of dividing a number by zero.

Program terminates abnormally.

Logical Errors

Problem in logic leads to wrong output or unwanted behaviour.

Like wrong condition in loop.

Exceptions

Exceptions are run-time abnormal conditions that a program encounters during its execution

There are two types of exceptions

① Synchronous

② Asynchronous (beyond the program's control, like disc failure)

Why exception Handling?

- Separation of error handling code from normal code, otherwise code is less readable and maintainable.
- A function can throw many exceptions, but may choose to handle some of them. The other exception which are thrown, but not caught can be handled by caller. If the caller chooses not to catch them then the exceptions are handled by the caller of the caller.

In C++, both basic types and object types can be thrown as exceptions (unlike Java)

Exception Handling

Exception handling consists of three keywords

- try
- throw
- catch

try

The try statement allows you to define a block of code to be tested for errors while it is being executed.

```
try
{


}
```

throw

The throw keyword throws an exception when a problem is detected, which lets us create a custom error.

There is no concept of automatic throw (unlike Java) in C++

catch

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs.

Example

```
void f1()
{
    int age, vote;
    try {
        cout<<"Enter your age";
        cin>>age;
        if(age<18)
          throw 1;
        cout<<"\nVote for your favourite actor";
        cout<<"\n1. Amir Khan";
        cout<<"\n2. Shahrukh Khan";
        cout<<"\n3. Akshay Kumar";
        cin>>vote;
        cout<<"\nThank You for your vote";
    }
    catch (int e){
       cout<<"You are not eligible to cast your vote";
    }
    Cout<<"\nEND";

}
```

Multiple throws

It is possible to write multiple throw statements in the same try block.

Multiple catch

To handle different exceptions in the same try block you can write multiple catch blocks.

catch all

There is a special catch block called 'catch all' `catch(...)` that can be used to catch all types of exceptions.

```
void f1()
{
    int age, vote;
    char nationality[20];
    try {
        cout<<"Enter your age";
        cin>>age;
        if(age<18)
          throw 1;
        cout<<"Enter your nationality";
        cin.ignore();
        cin.getline(nationality,20);
        if (strcmp(strupr(nationality),"INDIAN"))
            throw nationality;

        cout<<"\nVote for your favourite actor";
        cout<<"\n1. Amir Khan";
        cout<<"\n2. Shahrukh Khan";
        cout<<"\n3. Akshay Kumar";
        cin>>vote;
        if(vote>3||vote<1)
            throw 2.0;
        cout<<"\nThank You for your vote";
    }
catch(int e){
  cout<<"You are not eligible to cast your vote";
}
catch(char *n){
  cout<<"You have to be Indian";
}
catch(...){
  cout<<"Invalid Vote";
}
cout<<"\nEND";
}
```

No implicit Conversion

Implicit type conversion doesn't happen for primitive types.

Throwing a character constant will not converted implicitly for catch with int type receiver.

If an exception is thrown and not caught anywhere, the program terminates abnormally.

Example

```
void f2()
{
    int balance = 5000, amt;
    cout <<"Enter amount to withdraw";
    cin >> amt;
    if (amt > balance)
        throw 1;
    balance -=amt;
    cout<<"New balance is :"<<balance;
}
```

std:: terminate() is called by the C++ runtime when an exception is thrown and not caught

Defining own exception class

```
class MyException : public exception
{
    public:
        const char * what() const throw()
        {
            return "This is my custom error msg";
        }
};
```

```
void f1()
{
    int age, vote;
    char nationality[20];
    try {
        cout<<"Enter your age";
        cin>>age;
        if(age<18)
          throw *(new MyException());
        cout<<"Enter your nationality";
        cin.ignore();
        cin.getline(nationality,20);
        if (strcmp(strupr(nationality),"INDIAN"))
            throw nationality;

        cout<<"\n Vote for your favourite actor";
        cout<<"\n1. Amir Khan";
        cout<<"\n2. Shahrukh Khan";
        cout<<"\n3. Akshay Kumar";
        cin>>vote;

        if(vote>3||vote<1)
            throw 2.0;
        cout<<"\n Thank You for your vote";
    }
    catch (int e){
      cout<<"You are not eligible to cast your vote";
    }
    catch(char *n){
      cout<<"You have to be Indian";
    }
    catch(MyException &e){
        cout<<endl;
        cout<<e.what()<<endl;
    }

    catch(...){
      cout<<"Invalid Vote";
    }
    cout<<"\nEND";
}
```

A derived class exception should be caught before a base class exception

Like Java, C++ library has a Standard exception class which is base class for all exceptions

All objects thrown by components of the standard library are derived from exception class.

Therefore, all standard exceptions can be caught by catching exception type.