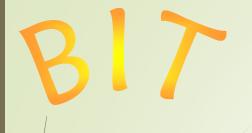


Exception Handling

Hu Sikang skhu@163.com



Contents

- Exception and Exception Handling
- Structure of exception handling in C++



Choices upon an error

- Ignore the error
- Terminate immediately
- Set an error flag, check by the caller
- Exception handling

817

Common Exceptions

out-of-bound array subscript

dividing by zero

```
double div(double x, double y)
{    return x / y; }
```

arithmetic overflow

```
int i = 1;
while (i > 0) i++;
```



Without Exception Handling

```
#include <iostream>
using namespace std;
double Div(double a, double b)
  if (b == 0)
   cout << "Untenable
           arguments to Div()";
   exit(0);
  return a / b;
```

```
int main()
   double x, y, z;
   cout << "Enter two numbers: ";
  cin >> x >> y;
  z = Div(x, y);
   cout << x << " / " << y <<
           " = " << z << endl;
  return 0;
```



What is Exception Handling?

• It is a mechanism that allows a calling program to detect and possibly recover from errors during execution.



With Exception Handling

```
#include <iostream>
using namespace std;
double Div(double a, double b)
{
   if (b == 0)
     throw "Divided by zero";
   return a / b;
}
```

```
int main() {
        double x, y, z;
        cout << "Enter two numbers: ";
        cin >> x >> y;
        try {
          z = Div(x, y);
           // The statement should NOT be
          // written after catch.
           cout << x << " / " << y << " = " << z;
        catch (const char* info)
        { cout << info << endl; }
        return 0;
```



Structure of Exception Handling

```
try {
     statement-list
catch (exception1) {
     statement-lis1t
catch (exception2) {
     statement-list2
catch (...) {
     statement-list...
```

From top to bottom



Passing Data with Exceptions

```
class CArray {
private: int* v, lower, upper;
public;
       CArray(int I, int u) : lower(I), upper(u)
               if (lower < 0) throw CError(0);</pre>
                if (upper < 0) throw CError(1);</pre>
                v = new int[upper - lower + 1];
                                                  int& CArray::operator[](int i) {
       int& operator[](int);
        ~CArray() { if (v) delete[] v; }
                                                          if (i \ge lower & i \le upper)
                                                                  return *(v + i - lower);
                                                          throw CError(2);
                                                  };
```



Passing Data with Exceptions

```
class CError {
private: int index;
public:
        CError(int i) { index = i; }
        string Info()
                switch (index)
                case 0:
                        return "Lower Error";
                                                 break;
                case 1:
                        return "Upper Error";
                                                 break;
                case 2:
                        return "Subscriptor Error"; break;
};
```



Passing Data with Exceptions

```
#include <iostream>
#include <string>
using namespace std;
int main()
       try
               CArray arr(0, 10); // CArray arr(-1, 10)
               arr[9] = 90; // arr[10] = 100
       catch (CError error)
               cout << error.Info() << endl;</pre>
       return 0;
```



Multiple Handlers

 Most programs performing exception handling have to handle more than one type of exception. A single try block can be followed by multiple handlers(catch), each configured to match a different exception type.

B int main()

Multiple Handlers

```
double Div(double a, double b) {
   if (b == 0)
   {     throw "Untenable arguments to Div() ";   }
   return a / b;
}
```



Exception with no Catch

- If no catch matches the exception generated by the try block, the search continues with the next enclosing try block.
- If no catch found, error!

```
void Callfun(const array& a)
{
    try

    fun (a);
}
// error if no catch!
}
```



Using Inheritance

```
#include <exception>
class exception;
class CMyException: public exception;
try {
 catch (CMyException& my) {
 catch (Exception) {
```



Using Inheritance

```
#include <iostream>
#include <exception>
using namespace std;
class CMyException: public exception
public:
      virtual const char* what() const throw()
            return "CMyException";
};
```

817

Using Inheritance

```
int main()
{
    try
    {
        throw CMyException();
    }
    catch (const CMyException& my) { cout << my.what() << endl; }
    catch (const exception& e) { cout << e.what() << endl; }
    return 0;
}</pre>
```