Object Oriented Programming with C++

2024 Spring Semester

21 CST H3Art

Chapter 14 Exception Handling（不完全总结，24spring考试没有涉及）

- **Exception（异常）**: An abnormal condition that occurs during program operation:
  - `new` operator cannot obtain the required memory **（内存分配时可用内存不足）**
  - data subscript is out of bounds **（下标越界）**
  - divisor `0` **（除以 `0`）**
  - invalid parameters **（无效参数）**
  - open non-existent files **（打开不存在的文件）**
- **Exception Handling（异常处理）**: to provide means to **detect** and **report** an "exceptional（异常的） situation" so that **appropriate action** can be taken.
- **Traditional exception handling methods（传统异常处理方式）**: Continuously test the necessary conditions for the program to continue running, and to handle the test results.
  - Example:

    ```
    int x,y,arr[10];

    cin >> x >> y;
    if (y == 0) cerr << "error";
      else cout << "x/y=" << x/y << endl;

    cin >> x;
    if (x > 9) cerr << "error";
      else cout << "arr[" << x << "]=" << arr[x] << endl;
    ```

  - Traditional method makes **error handling code distributed in various parts of the whole program**, which makes the program "polluted" by the error handling code and becomes **diffcult to read（传统异常处理方法，大量异常处理代码四处分布，程序难以阅读）**.
- **Exception Handling Mechanism（异常处理机制）**
  - C++ Exception Handling: the basic idea is to **put the exception occurrence and exception handling in different functions（将异常发起和异常处理置于不同的函数中）**.
  - Three **keywords（关键字）**: `try`, `throw` and `catch`:
    - `try` block: Detect and throws an exception
    - `catch` block: Catch and handles the exception
  - Example:

```
try{ //try block
  ...
  if err_1 throw xx_1
  ...
  if err_2 throw xx_2
  ...
  if err_n throw xx_n
}
catch(type1 arg){...}
catch(type2 arg){...}
catch(typem arg){...}
```

- Exceptions can be **thrown by functions**:

```
void divided(int x, int y, int z) {
  if((x-y) != 0)
      cout << "Result=" << z / (x - y) << endl;
  else
      throw(x - y);
}

int main() {
  try {
    cout << "we are inside the try block" << endl;
    divide(10, 20, 30);
    divide(10, 10, 20);
  } catch (int i) {
    cout << "Caught the exception" << endl;
  }
  return 0;
}
```

- **The execution logic of exception handling（异常处理的执行逻辑）**
  - When a `try` block is encountered during program execution, it will **enter the `try` block and execute the statements** in it in the **normal program logic order**.
  - If all statements in the `try` block are executed **normally without any exception**, no exception will be thrown in the `try` block. In this case, the program will **ignore all `catch` blocks and continue to execute program** statements other than `catch` blocks in sequence.
  - If an **error occurs** in a statement during the execution of a `try` block and an exception is thrown, the program **control flow will transfer to the `catch` block（控制流进入 `catch` 块）**, and **all statements after the throw statement in the `try` block will not be executed（`try` 块中触发异常的语句之后的所有语句不会再运行）**.
  - C++ will compare the data type of the exception with the data type specified in each `catch` parameter table in the order in which the `catch` blocks appear. As long as **one `catch` block catches the exception, the remaining `catch` blocks will be ignored（`catch` 块会根据抛出的异常类型自动捕获异常，当异常被其中一个 `catch` 块捕获，其他的 `catch` 块不会再生效）**.
    - Example:

```cpp
#include <iostream>

using namespace std;

int main() {
  cout << "1--befroe try block..." << endl;
  try {
      cout << "2--Inside try block..." << endl;
      throw 10;
      cout << "3--After throw ...." << endl;
  } catch (int i) {
      cout << "4--In catch block1 ... exception ... errcode is ... " << i << endl;
  } catch (char *s) {
      cout << "5--In catch block2 ... exception ... errcode is ... " << s << endl;
  }
  cout << "6--After Catch..." << endl;

  return 0;
}
```

Output:

```
1--befroe try block...
2--Inside try block...
4--In catch block1 ... exception ... errcode is ... 10
6--After Catch...
```

  - If no `catch` can match the exception, C++ will call the system's **default exception handler** to handle the exception, and its usual practice is to **directly terminate the program（系统默认的异常处理是终止程序）**.
- **Handling exceptions in a function（在函数中处理异常）**
  - Exception handling can be **localized to a function**, that is, the `try` - `throw` - `catch` structure for handling exceptions is placed in the function, and **each time the function is called, the exception will be reset（每次函数调用时，异常会被重置）**.
  - Example:

```
#include <iostream>

using namespace std;

void test(int x) {
  try {
    if (x == 1)
      throw x;
    else if (x == 0)
      throw 'x';
    else if (x == -1)
      throw 1.0;
    cout << "End of try block" << endl;
  } catch (char c) {
    cout << "Caught a character" << endl;
  } catch (int m) {
    cout << "Caught an integer" << endl;
  } catch (double d) {
    cout << "Caught a double" << endl;
  }
  cout << "End of try-catch system\n" << endl;
}

int main() {
  test(0);
  test(-1);
  test(2);
  return 0;
}
```

Output:

```
Caught a character
End of try-catch system

Caught a double
End of try-catch system

End of try block
End of try-catch system
```

- Putting the program code that **generates exceptions in one function（产生异常的代码在一个函数中）** and the function code that **detects and handles exceptions in another function（检测和处理异常的代码在另一个函数中）** can make exception handling **more flexible and practical（更灵活更实用地处理异常）**.
  - Example:

```cpp
#include <iostream>

using namespace std;

void temperature(int t) {
    if (t == 100)
      throw "沸点！";
    else if (t == 0)
      throw "冰点！";
    else
      cout << "ok" << endl;
}

int main() {
  try {
    temperature(0);
    temperature(10);
    temperature(100);
  } catch (char const *s) {
    cout << s << endl;
  }
  return 0;
}
```

中间新插入了异常的匹配细节，包括：只有const转换、基类匹配派生类、指针匹配数组名这三种方式可用于异常匹配，以及默认异常处理会将程序终止

- **Catch all exceptions（捕获所有异常）**
  - In most cases, catch is only used to catch a specific type of exception, but it also has the ability to catch all exceptions.
  - **Its form is as** `catch(...){}`.
  - Example:

    ```cpp
    #include <iostream>

    using namespace std;

    void Errhandler(int n) {
      try {
        if(n==1) throw n;
        if(n==2) throw "dx";
        if(n==3) throw 1.1;
      } catch(...) {
        cout<<"catch an exception..."<<endl;
      }
    }

    int main() {
      Errhandler(1);
      Errhandler(2);
      Errhandler(3);
      return 0;
    }
    ```

- **Rethrowing an exception（异常重抛出）**

- If the catch block cannot handle the captured exception, or can only handle part of the exception, the rest of the exception needs to be processed by its outer calling function. It can **use the throw statement without any parameters to throw the exception again**.
- Example:

```cpp
#include <iostream>

using namespace std;

void Errhandler(int n) {
  try {
    if(n==1) throw n;
    cout << "all is ok..." << endl;
  } catch(int n) {
    cout << "catch an int exception inside..." << n << endl;
    throw; //再次抛出本catch捕获的异常
  }
}

int main() {
  try {
    Errhandler(1);
  } catch(int x) {
    cout << "catch int an exception in main..." << x << endl;
  }
  cout << "...End..." << endl;
}
```