C++

程式語言（二）

Introduction to Programming (II)

Exception Handling

Joseph Chuang-Chieh Lin

Dept. CSE, NTOU

Platform/IDE

- Dev-C++

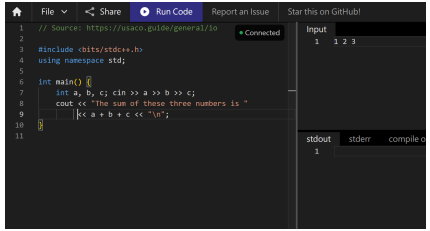
https://www.pngegg.com/en/search?q=Dev-C

- Codeblocks

https://icons8.com/icons/set/code-blocks

- OnlineGDB (https://www.onlinegdb.com/)

- Real-Time Collaborative Online IDE (https://ide.usaco.guide/)



Textbooks (We focusing on C++11)

- ***Learn C++ Programming by Refactoring* (由重構學習 C++ 程式設計). Pang-Feng Liu (劉邦鋒). NTU Press. 2023.**
- ***C++ Primer. 5th Edition.* Stanley B. Lippman, Josée Lajoie, Barbara E. Moo. 2019.**
- *Effective C++*. Scott Meyers. O'Reilly. 2016.
- *Thinking in C++. Vol. 1: Introducing to Standard C++*. 2nd Edition. Bruce Eckel. Prentice Hall PTR. 2000.

Useful Resources

- Tutorialspoint
  - https://www.tutorialspoint.com/cplusplus/index.htm
  - Online C++ Compiler
- Programiz
  - https://www.programiz.com/cpp-programming
- LEARN C++
  - https://www.learncpp.com/
- MIT OpenCourseWare - Introduction to C++
  - https://ocw.mit.edu/courses/6-096-introduction-to-c-january-iap-2011/pages/lecture-notes/
- Learning C++ Programming
  - https://www.programiz.com/cpp-programming
- GeeksforGeeks
  - https://www.geeksforgeeks.org/c-plus-plus/

Exception Handling

Motivation

- Error handling has been one of the most difficult issues since the beginning of programming languages.
- It's difficult to design a good handling scheme.
- Exception handling wires errors **directly into** the programming language and sometimes even the operating system.
- It's as if parallel path of execution can be taken when something goes wrong.
- Instead of just exiting the program, we are able to set things right and restore the execution of a program.
  - This makes the system **more robust**.

Examples for why it is necessary

```
#include <iostream>
using namespace std;

int divide(int a, int b) {
    return a / b;
    // OH NO...
    //If b == 0, program crashes
}

int main() {
    int result = divide(10, 0);
    // Crash!
    cout << "Result: "
         << result << endl;
    cout << divide(10, 2);
    return 0;
}
```

```
#include <iostream>
#include <stdexcept>
using namespace std;

int divide(int a, int b) {
    if (b == 0)
        throw runtime_error("Orz: Division by zero!");
    return a / b;
}

int main() {
    try {
        int result = divide(10, 0);
        // Throws an exception
        cout << "Result: " << result << "\n";
    } catch (const exception& e) {
        cout << "Exception Caught: " << e.what()
             << endl;
    }
    cout << divide(10, 2);
    return 0;
}
```

- Prevent unexpected runtime errors which may cause a program to terminate abruptly.

https://onlinegdb.com/2m7XbO_-B

Example: Multiple Exception Types

https://onlinegdb.com/0dDCuWBzg

```
#include <iostream>
#include <string>
#include <stdexcept>
using std::endl;
using std::cout;

void testFunc(int code) {
     if (code == 1)
          throw std::runtime_error("Runtime error occurred!");
     if (code == 2) throw 100;  // Throwing an integer
     else throw std::invalid_argument("Invalid argument!");
}
int main() {
    try {
        testFunction(3);
    } catch (const std::runtime_error& e) {
        cout << "Caught runtime error: " << e.what() << endl;
    } catch (int e) {
        cout << "Caught an integer exception: " << e << endl;
    } catch (const std::invalid_argument& i) {
        cout << "Caught an invalid argument: " << i.what() << endl;
    }
    return 0;
}
```

Example: Generic Exception

```
#include <iostream>
#include <stdexcept>

int main() {
    try {
        throw std::exception(); // Throwing a generic exception
    } catch (const std::exception& e) {
        std::cout << "Exception caught: " << e.what() << std::endl;
       // note that here what() is virtual
    }
    return 0;
}
```

Example: Logic Error Exception

```
#include <iostream>
#include <stdexcept>
#include <vector>
Using std::endl;

int main() {
    try {
        std::vector<int> vec = {1, 2, 3};
        std::cout << vec.at(5);  // accessing out-of-range index
       // an exception is thrown automatically by std::vector::at()
    } catch (const std::out_of_range& e) {
        std::cout << "Out of Range Exception: " << e.what() << endl;
    }
    return 0;
}
```

Example: Memory Allocation Error Exception

```
#include <iostream>
#include <stdexcept>

int main() {
    try {
        int* arr = new int[100000000000]; // Too large, allocation fails
        // note that operator 'new' automatically throws std::bad_alloc
    } catch (const std::bad_alloc& e) {
         std::cout << "Memory Allocation Failed: " << e.what() << "\n";
    }
    return 0;
}
```

Custom Exception Classes

Example:
Derived from `std::exception`

```
#include <iostream>
#include <exception> // for base exception class only

class MyException : public std::exception { // Custom exception class
public:
    const char* what() const noexcept override {
        return "Custom exception occurred!";
    }
};

int main() {
    try {
        throw MyException();  // Throwing custom exception
    } catch (const MyException& e) {
        std::cout << "Caught exception: " << e.what() << std::endl;
    }
    return 0;
}
```

Example:
A Custom Exception Class (from Prof. Liu's textbook)

https://github.com/pangfengliu/Cplusplus-refactor/blob/main/operator/rational-io-throw.cc

```
class NoSlash: public exception {
    virtual const char* what() const noexcept { // override what()
        return "No slash found";
    }
};
NoSlash noSlash;

istream& operator>>(istream &in, Rational &r) {
  string s;
  if (in >> s) {
    auto slash {s.find('/')};
    if (slash == string::npos)
      throw noSlash;
    int q = stoi(s.substr(0, slash));
    int p = stoi(s.substr(slash + 1));
    r = Rational(q, p);
  }
  return in;
}
```

```
#ifndef RATIONAL_H
#define RATIONAL_H
#include <string>

class Rational {
 private:
  int p, q;     // q/p
  void simplify();
 public:
 Rational(int b = 0, int a = 1);
  Rational operator+(const Rational &r) const;
  Rational operator-(const Rational &r) const;
  Rational operator*(const Rational &r) const;
  Rational operator/(const Rational &r) const;
  bool operator<(const Rational &r) const;
  bool operator==(const Rational &r) const;
  bool operator>(const Rational &r) const;
  void print(string msg = "", ostream &out = cout) const;
};
#endif
```

Discussions & Questions