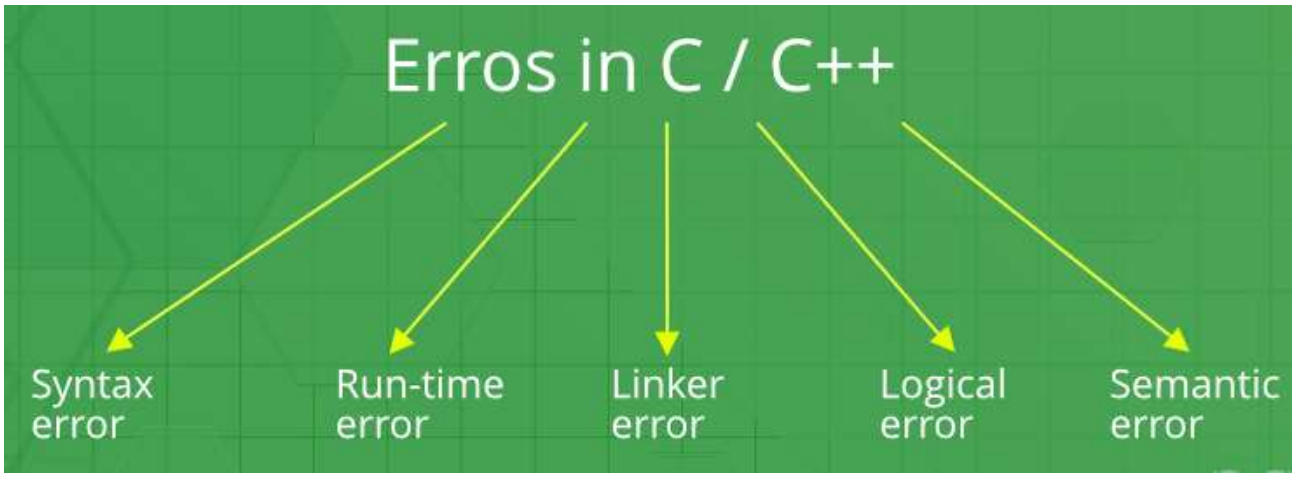
หน่วยที่ 6

การแก้ไขข้อผิดผลาดของโปรแกรม



หัวข้อเรื่อง

6.1 ความผิดผลาดของการเขียนโปรแกรม

6.2 การทำงานของ try / catch block

การพัฒนาโปรแกรม อาจมีข้อผิดพลาดที่ไม่ถูกต้องเกิดขึ้นกับคำสั่งที่เขียน เช่น หาทรัพยากรที่เรียกใช้ไม่พบ หรือการทำงานอยู่นอกเหนือข้อจำกัดที่กำหนด ฯลฯ ฉะนั้นการเขียนโปรแกรมที่ดีจะต้องมีการจัดการข้อผิดพลาดที่เกิดขึ้นมาได้

การจัดการข้อผิดพลาด (Exceptions Handling) ที่จะอธิบายต่อไปนี้ เป็นข้อกำหนดใหม่ที่เสนอโดยมาตรฐาน ANSI C++ หาก C++ คอมไพเลอร์ที่ใช้อยู่ไม่สนับสนุนมาตรฐานนี้ก็ไม่อาจใช้ข้อกำหนดนี้ได้

ในบทนี้จะกล่าวถึงชุดคำสั่งจัดการข้อผิดพลาดของโปรแกรม โดยควบคุมข้อผิดพลาดในการทำงานด้วยคำสั่ง try, throw และ catch รวมไปถึงไลบรารีมาตรฐานที่เกี่ยวกับการจัดการข้อผิดพลาดของ C++

6.1 ความผิดพลาดของการเขียนโปรแกรม

โดยทั่วไปแล้วอาจแบ่งชนิดของความผิดพลาดที่เกิดขึ้นจากการเขียนโปรแกรม ได้เป็น 2ประเภท คือ ความผิดพลาดขณะคอมไพล์หรือที่เรียกว่า Compile-Time Errors อาจเรียกได้อีกอย่างว่าSyntax Error และความผิดพลาดขณะทำการประมวลผล หรือ Run-Time-Errors

1. ความผิดพลาดขณะคอมไพล์ มักเกิดจากการเขียนโปรแกรมไม่ทำตามรูปแบบคำสั่ง เช่น ลืมใส่ {} () เมื่อจบประโยค เป็นต้น ลักษณะความผิดพลาดแบบนี้จะคุ้นเคยกันเพราะเป็นประสบการณ์พื้นฐานที่ต้องแก้ไขโปรแกรม ไม่เช่นนั้นโปรแกรมก็จะไม่ทำงาน

2. ความผิดพลาดขณะทำการประมวลผล อาจจำแนกได้เป็น 2 สาเหตุใหญ่ ๆ คือ ความผิดพลาดจากวิธีการคิด (Program Logic) และความผิดพลาดที่เกิดจากสภาวะทำงานไม่สมประกอบ

2.1 ความผิดพลาดจากวิธีการคิด การเขียนโปรแกรมจำเป็นจะต้องป้องกันโดยคิดหาวิธีการตรวจจับความผิดพลาดดังกล่าว ในภาษา C++ โดยใช้คำสั่ง try, throw และ catch

2.2 ความผิดพลาดที่เกิดจากสภาวะการทำงานของระบบ เช่น เน็ตเวิร์กไม่ทำงานดิสก์เต็มเป็นต้น

การจัดการข้อผิดพลาดจากวิธีการคิด

ข้อผิดพลาดที่เกิดขึ้นนี้ต้องทำการจัดการให้ได้ และ ภาษา C++ ได้รวบรวมเครื่องมือที่จะจัดการกับข้อผิดพลาดโดยทันทีไว้ 3 คำสั่ง คือ try, throw และ catch

รูปทั่วไปของคำสั่ง

```
try {
        // statements
}
catch(dataType1 identifier) {
        // exception handling code
}
        ::
        ::
catch(dataTypen identifier) {
        // exception handling code
}
catch(...) {
        // exception handling code
}
```

ภายในส่วนของคำสั่ง try ประกอบด้วยกลุ่มคำสั่งที่อาจทำให้เกิดข้อผิดพลาดและกรณีที่มีข้อผิดพลาดเกิดขึ้น คำสั่งในส่วนของ try จะไม่ได้ประมวลผล ส่วนคำสั่ง catch ประกอบด้วยการระบุชนิดของข้อผิดพลาดที่เกิดขึ้นในแต่ละกรณี และภายใน block จะประกอบด้วย ชุดคำสั่งที่จะจัดการกับข้อผิดผลาดที่เกิดขึ้น

6.2 การทำงานของ try / catch block

ถ้าไม่มีข้อผิดพลาดเกิดขึ้นภายในประโยค try ประโยค catch ทั้งหมดก็จะถูกมองข้ามไปโปรแกรมก็จะประมวลผลต่อจากประโยค catch ประโยคสุดท้าย

ถ้าเกิดข้อผิดพลาดขึ้นภายในประโยคของ try ณ จุดใดจุดหนึ่ง ประโยคคำสั่งที่เหลือก็จะถูกยกเลิกการทำงาน โปรแกรมจะค้นหาประโยค catch ตามลำดับ โดยค้นหาประโยค catch ที่มีการนิยามไว้สอดคล้องกับ

ข้อผิดพลาดที่เกิดขึ้น โดยที่ถ้าข้อผิดพลาดตรงกับพารามิเตอร์ของประโยค catch ใด ประโยค catch นั้นก็จะทำงาน ประโยค catch อื่น ๆ ถูกมองข้ามไป ไม่มีการประมวลผลใด ๆ

ประโยค catch ประโยคสุดท้ายที่มีจุดสามจุดในวงเล็บเป็นประโยค catch สำหรับตรวจจับข้อผิดผลาดทั่ว ๆ ไป(อื่น)

ลองพิจารณาประโยค catch ต่อไปนี้

```
catch (int x) {
// ecception handling code
}
```

ตัวแปร x คือ พารามิเตอร์ของประโยค catch นี้

ส่วนชนิดของข้อมูล int ของพารามิเตอร์ x เป็นการระบุว่าประโยค catch นี้จะสามารถตรวจจับข้อผิดพลาดที่มีชนิดข้อมูลเป็นเลขจำนวนเต็ม

ประโยค catch แต่ละประโยคสามารถมีพารามิเตอร์ได้อย่างมากที่สุดเพียง 1 ตัว เท่านั้น

6.2.1 ลำดับของคำสั่ง catch

คำสั่ง catch สามารถตรวจจับข้อผิดพลาดที่มีการกำหนดชนิดของข้อผิดพลาดไว้อย่างเฉพาะเจาะจงหรือสามารถตรวจจับข้อผิดพลาด โดยทุกชนิด

ตรงส่วนหัวของประโยค catch จะเป็นการกำหนดชนิดข้อผิดพลาดและถ้าประโยค catch กำหนดเป็นจุดสามจุด (...) แล้วจะเป็นการทำให้ประโยค catch นั้นตรวจจับผิดพลาดทุกชนิด เนื่องจาก ประโยค catch ที่ตรงส่วนหัวกำหนดเป็นจุดสามจุด (...) สามารถตรวจจับข้อผิดพลาดทุกประเภท ดังนั้น ประโยค catch ควรจะเป็นประโยค catch ประโยคสุดท้าย เพราะถ้านำประโยค catch ตามด้วยจุดสาม จุดนี้ไปไว้ก่อนประโยค catch อื่น ๆ จะทำให้ประโยค catch อื่นๆ ที่ตามหลังประโยค catch ที่มีจุดสาม จุด จะไม่มีโอกาสได้รับการประมวลผลตราพับข้อผิดพลาดใดๆเลย ตัวอย่างเช่นถ้ากำหนดให้ประโยค try ตามด้วยประโยค catch ดังต่อไปนี้

```
try {
```

```
        //ประโยคคำสั่งต่างๆ
}
catch (...) {                    // ประโยค catch ที่ 1
        //ประโยคคำสั่งต่าง ๆ
}
catch (int x) {                  // ประโยค catch ที่ 2
        //ประโยคคำสั่งต่าง ๆ
}
```

จากประโยคคำสั่ง try และ catch ข้างต้น ประโยค catch ที่ 2 catch (int x) จะไม่มีโอกาสได้รับการประมวลผล แม้กระทั่งเพียงครั้งเดียว ประโยค catch นี้เป็นเสมือนประโยคคำสั่งที่ไม่จำเป็นต้องมีเพราะไม่มีโอกาสประมวลผลใด ๆ เลย

การโยนข้อผิดพลาด (Throwing on Exception)

โดยปกติการเขียนโปรแกรม จะมีการวางเงื่อนไขต่าง ๆ ภายในคำสั่งเพื่อตรวจจับข้อผิดพลาดที่เกิดขึ้น ในกรณีที่เกิดข้อผิดพลาดขึ้นมา ก็จะต้องให้โปรแกรมดำเนินการกับข้อผิดพลาดที่เกิดขึ้นนั้นตามคำสั่งที่ได้เขียนขึ้นมา เรียกว่า การโยนข้อผิดพลาด (throwing on exception) ซึ่งทำงานได้โดยคำสั่ง throw

เมื่อคำสั่งภายในประโยค try เกิดข้อผิดพลาดขึ้นและมีการตรวจจับขังผิดพลาด ด้วยประโยค catch ข้อผิดพลาดที่เกิดขึ้นจำเป็นต้องถูกดำเนินการ โดยใช้คำสั่ง throw รูปแบบคำสั่ง throw โดยทั่วไป คือ

```
throw exception ;
```

คำสั่ง throw จะตามด้วยนิพจน์ซึ่งอาจเป็นค่าคงที่ ตัวแปรหรืออ็อบเจ็กต์

การใช้คำสั่ง try และ catch ในโปรแกรม

ตัวอย่างที่ 6.6 แสดงการทำงานของ try throw และ catch

```
#include <iostream>
```

```
using namespace std;
int main () {
    char myarray[10];
    try
        {
        for (int n=0; n<=10; n++)
        {
            if (n>9) throw "Out of range";
            myarray[n]='z';
        }
    }
    catch(const char *str)
    {
    cout << "Exception: " << str << endl;
    }
    return 0; }
```

ผลลัพธ์

```
Exception: Out of range
```

จากตัวอย่าง เป็นโปรแกรมการทำงานซ้ำ และมีการตรวจสอบเงื่อนไข กรณีเมื่อ n มีค่ามากกว่า 9 ให้ทำการโยนข้อผิดพลาดที่เกิดขึ้นก่อนที่นำไปเก็บไว้ในอาร์เรย์ myarray อาร์เรย์ myarray ก็จะมีค่า (0-9) เมื่อคำสั่ง throw ทำงาน การทำงานในส่วนของ try บล็อกก็จะสิ้นสุดลง หลังจากนั้นการทำงานจะไปยัง catch

บล็อก (ซึ่งจะทำงานเมื่อเกิดข้อผิดพลาดขึ้นเท่านั้น) เมื่อเสร็จแล้วโปรแกรมก็จะทำงานต่อไปในที่นี้ คือ return 0;

โดย catch จะรับเฉพาะพารามิเตอร์ที่ชนิดตรงกันเท่านั้น บ่อยครั้งที่สามารถมี catch หลายตัว (Overload) ซึ่งจะรับชนิดพารามิเตอร์ที่ต่าง ๆ กัน ในกรณีนี้จะมี catch บล็อกที่เข้ากันกับชนิดของข้อผิดพลาดเท่านั้นทำงาน (ชนิดพารามิเตอร์ของ throw ที่โยนมาให้)

ตัวอย่างที่ 6.7 แสดงการทำงานของ catch หลายตัว

```
#include <iostream>
using namespace std;
int main () {
    try
    {
    char * mystring;
    mystring = new char [10];
    if (mystring == NULL) throw "Allocation failure";
    for (int n=0; n<=100; n++)
        {
        if (n>9) throw n;
        mystring[n]='z';
        }
    }
catch (int i)
```

```
    {
    cout << "Exception: ";
    cout << "index " << i << " is out of range" << endl;
    }
catch (const char * str)
    {
    cout << "Exception: " << str << endl;
    }
return 0; }
```

ผลลัพธ์

Exception: index 10 is out of range

ในกรณีนี้จะมีความเป็นไปได้อย่างน้อย 2 กรณี ที่สามารถเกิดขึ้นได้ คือ

1. หากการร้องขอขนาดข้อความ 10 ตัวอักษร ไม่สามารถให้ได้ คือ mystring == NULL กรณีนี้ข้อผิดพลาดจะถูกโยนผ่านไปสู่ catch (const char * str)

2. หากตัวชี้ของ mystring เป็นคำที่กำหนด กรณีนี้ข้อผิดพลาดจะถูกโยนผ่านไปสู่ catch (int i) ซึ่งตรงกับพารามิเตอร์ที่เป็นเลขจำนวนเต็ม

5.6 ข้อผิดพลาดมาตรฐาน (Standard Exception)

ภาษา C++ มีการสร้างคลาสข้อผิดพลาด (Exception Class) เพื่อจัดการกับข้อผิดพลาดต่าง ๆ คลาสข้อผิดพลาดที่มีอยู่ในภาษา C++ นั้นเป็นคลาส พื้นฐานที่ถูกสร้างไว้สำหรับตรวจจับข้อผิดพลาดในการทำงานของโปรแกรม คลาสข้อผิดพลาดเหล่านี้จะมีสมาชิกฟังก์ชัน what สมาชิกฟังก์ชัน what มีหน้าที่ส่งข้อความสำหรับข้อผิดพลาดต่าง ๆ ที่เกิดขึ้น

การกำหนดส่วนหัวข้อโปรแกรมในการจัดการข้อผิดพลาด (Header Files for Exception Classes)

1. ข้อผิดพลาดพื้นฐาน ฐานและ bad_exception ใช้ <exception>

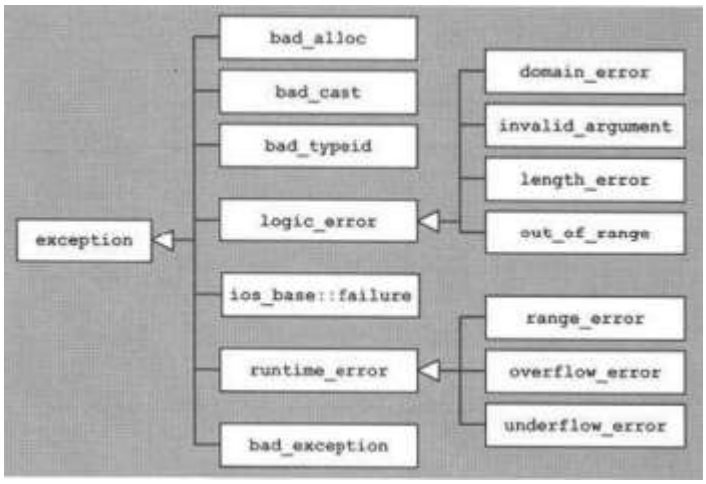
2. ข้อผิดพลาด bad_alloc ใช้ <new>

3. ข้อผิดพลาด bad_cast และ bad_typeid ใช้ <typeinfo>

4. ข้อผิดพลาด ios_base::failure ใช้ <ios>

5. ข้อผิดพลาดอื่น ๆ ใช้ <stdexcept>.

แสดงข้อผิดพลาดมาตรฐาน (Standard Exception)



ในภาษา C++ มีคลาสข้อผิดพลาด 2 คลาสที่ถูกสร้างขึ้นจากคลาสข้อผิดพลาดพื้นฐานสำหรับจัดการกับข้อผิดพลาดที่สำคัญต่าง ๆ คือ logic-error และ runtime-error คลาสทั้ง 2 คลาสนี้อยู่ในไฟล์ Header ชื่อ stdexcept

คลาส logic-error เป็นคลาสข้อผิดพลาดพื้นฐานที่จะจัดการกับข้อผิดพลาดทางด้านตรรกะ เช่น ข้อผิดพลาดเกี่ยวข้องกับ out-of-range หรือ การส่งพารามิเตอร์ไม่ถูกต้อง ตอนเรียกฟังก์ชันเป็น ข้อผิดพลาดเกี่ยวข้องกับคลาส invalid argument สตริงอ๊อบเจ็กต์มีขนาดที่กำหนดไว้แน่นอน ถ้าสตริงอ๊อบเจ็กต์ถูกใช้เก็บตัวอักษรเกินกว่าที่กำหนด ข้อผิดพลาด length-error จะเกิดขึ้น ซึ่งเราสามารถ เรียกใช้คลาสข้อผิดพลาด length-error ได้

คลาส runtime-error เป็นคลาสข้อผิดพลาดพื้นฐานที่จัดการกับข้อผิดพาดที่เกิดขึ้นในระหว่างการประมวลผลโปรแกรมเท่านั้น เช่น ข้อผิดพลาดที่เกิดจากการประมวลผล stack อาจทำให้เกิดข้อผิดพลาดที่

เรียกว่า underflow หรือ overflow ซึ่งเราก็สามารถเรียกใช้คลาสข้อผิดพลาด underflow และ overflow ได้ (เป็นคลาสที่สร้างต่อยอดมาจากคลาสพื้นฐาน runtime-error)

ตัวอย่างที่ 6.8 แสดงการทำงาน standard exceptions

```
#include <iostream>
#include < stdexcept >
using namespace std;
class myexception: public exception
{
    virtual const char* what() const throw()
    { return "My exception happened"; }
} myex;
int main () {
    try
    {
    throw myex;
    }
    catch (exception& e)
    {
    cout << e.what() << endl;
    }
return 0; }
```

ตัวอย่างที่ 6.9 แสดงการใช้คลาสข้อผิดผลาด bad-alloc

```
#include <iostream>
#include <new>
using namespace std;
#define SIZE 100000000
int main() {
    int *list[50];
    try {
        for(int i =0; i<50 ; i++) {
            list[i] = new int [SIZE];
            cout << "List # " << i << "of " << SIZE << "Components " << endl;
            }
    }
    catch (bad_alloc& objexh) {
        cout << "In the catch block of bad_alloc , exception : "
            << objexh.what() << endl;
    }
return 0;
}
```

สรุป

เทมเพลต คือ การนำเอาฟังก์ชันหรือคลาสมาเพิ่มความสามารถ ให้ทำงานกับข้อมูลได้หลากหลายชนิด โดยอาศัย พารามิเตอร์เทมเพลต (Template Parameters) ซึ่งเป็นตำแหน่งที่วางสำหรับชนิดข้อมูลและคลาสต่าง ๆ ฟังก์ชันเทมเพลตที่ใช้กำหนดชนิดข้อมูลของฟังก์ชัน มีรูปแบบคำสั่ง

```
template <class identifier> function _declaration;
```

การประกาศคลาสเทมเพลต จะมีรูปแบบ

```
template <class T,.... Class x{...};
```

การจัดการข้อผิดพลาด ในขณะการพัฒนาโปรแกรม จะใช้คำสั่ง try/catch เพื่อตรวจจับ โดยใช้ร่วมกับการจัดการข้อผิดพลาด (exception) รูปแบบทั่วไปของคำสั่ง

```
try { // //ประโยคคำสั่งต่างๆ
catch(dataType1 identifier) {// exception handling code }
```