كلية الهندسة – قسم المعلوماتية

مقرر برمجة 2

د. علي سليمان

محاضرة العاشره

Exception Handling

الفصل الاول 2024-2023

Exception Handling

معالجة الاستثناءات

1. introduction.
2. Exception Handling Fundamentals.
3. Catching Class Types.
4. Handling Derived-Class Exceptions.
5. Exception Handling Options.
6. Understanding terminate() and unexpected().
7. Setting the Terminate and Unexpected Handlers.
8. The uncaught_exception Function().
9. Case Study.

1. مقدمة.
2. المفاهيم الأساسية لمعالجة الاستثناءات.
3. التقاط الأصناف.
4. التعامل مع استثناءات من أصناف مشتقة.
5. خيارات معالجة الاستثناءات.
6. فهم terminate() و unexpected().
7. تخصيص معالجات terminate و unexpected.
8. التابع uncaught_exception().
9. دراسة حالة.

المحاضرة من المراجع :

[1]- Deitel & Deitel, C++ How to Program, Pearson; 10th Edition (February 29, 2016)

[2]- د.علي سليمان, البرمجة غرضية التوجه في لغة C++ 2009-2010

Introduction

6-1- مقدمة

1. تتيح معالجة الاستثناءات exception handling إدارة الأخطاء في وقت التنفيذ run-time errors بطريقة مخصصة.
2. يمكن للبرنامج باستخدام معالجة الاستثناءات أن يقوم تلقائياً بتنفيذ إجرائية معالجة خطأ عند حدوثه.
3. الفائدة الأساسية من معالجة الاستثناءات أنها تؤتمت الكثير من مقاطع شيفرة معالجة الخطأ التي كان من الواجب كتابتها سابقاً بشكل مباشر في أي برنامج ضخم.
4. يصبح البرنامج أكثر وضوحاً ومتانة ومقاومة للأعطال.
5. والغاية هى الإحاطة بأخطاء البرنامج ومعالجتها بدلاً من الوقوع بها والمعاناة من نتائج حدوثها حيث ينتهي البرنامج إذا لم يقم المبرمج بمعالجة هذه الأخطاء باستخدام اسلوب معالجة الاستثناءات.

6- 1- المفاهيم الأساسية لمعالجة الاستثناءات

- بعض الاحتمالات التي تتطلب معالجة الاستثناءات:
  - ✓ إذا كان البرنامج يعالج مهام حساسة وسيتم طرحة في السوق وسيستفيد منه شريحة عريضة لابد من العناية بمعالجة الاستثناءات.
  - ✓ عند إعطاء قيمة لدليل الأخطاء، حيث لن يعبر هذا الدليل عن كامل أماكن حدوث الخطأ له.
  - ✓ يعود تقدير المعالجة من غيرها إلى المبرمج وتقييمة لأهمية عدم انقطاع البرنامج وسرعته.
  - ✓ إذا كان معدل حدوث الخطاء قليل جداً لتنفيذ تعليمة ما يمكن تجاهل المعالجة وبالتالي سيتم قطع البرنامج مع عدم الانتباه إلى تحرير المصادر في حال تم اشغالها.
  - ✓ فحص الشروط التي يحدث عندها الخطأ وإعطاء رسالة ثم استدعاء التابع exit() للخروج من البرنامج وفق رقم خطأ معين.
- عيوبه:
  - ✓ معالجة الاستثناءات تسبب بطء في البرنامج.

6- 1- المفاهيم الأساسية لمعالجة الاستثناءات

- تم بناء أسلوب معالجة الاستثناءات في لغة C++ باستخدام ثلاث تعليمات (كلمات مفتاحية) وهي:

```
try
{}
throw args ;
catch  (data Type args )
{}
```

- بشكل عام، يتم تضمين التعليمات التي نريد مراقبتها إن كانت تولد استثناء ضمن الكتلة try { }. في حال حصول خطأ أو استثناء ضمن الكتلة try يتم قذفه باستخدام throw ومن ثم التقاطه باستخدام كتلة catch { } التالية لكتلة try ومن ثم معالجته.
- يتم تنفيذ الشيفرة المراد مراقبتها من أجل الاستثناءات ضمن المقطع try (التوابع التي يتم استدعاؤها ضمن المقطع try يمكن أيضاً أن تولد استثناءات)، يتم قذف استثناء من قبل الشيفرة التي تتم مراقبتها والتقاطه من قبل الكتلة catch والتي تتبع مباشرة الكتلة try التي تم قذف الاستثناء ضمنها.
- وتكون الصيغة العامة لـ try و catch كما يلي :

6- 1- المفاهيم الأساسية لمعالجة الاستثناءات

```
try {
// try block
}
catch (type1 arg) {
// catch block
}
catch (type2 arg) {
// catch block
}
catch (type3 arg) {
// catch block
}
..
..
catch (typeN arg) {
// catch block
}
```

- يمكن للمقطع try أن يكون قصيراً (عدة تعليمات) ضمن تابع واحد، ويمكن أن يشمل وضع كامل تعليمات تابع main() ضمن المقطع try الأمر الذي يؤدي لجعل كامل البرنامج تحت المراقبة.
- عند قذف استثناء ما، يتم التقاطه من قبل التعليمة catch المقابلة (المعنيه)، والتي تقوم بمعالجته.
- يمكن أن يكون هناك أكثر من عبارة catch مرفقة مع مقطع try ويتم تحديد أي تعليمة catch سيتم تنفيذها بحسب نوع الاستثناء الحاصل. أي، إذا كان نوع البيانات المحدد في التعليمة catch يتطابق مع ذلك الخاص بالاستثناء الملقى، عندئذ يتم تنفيذ تلك التعليمة catch وتجاهل باقي التعليمات الأخرى.

6- 1- المفاهيم الأساسية لمعالجة الاستثناءات

- عند التقاط استثناء، يتلقى **args** قيمته، يمكن التقاط أي نوع من البيانات بما في ذلك الأصناف التي تقوم بإنشائها. وإذا لم يتم قذف أي استثناء، فذلك يعني عدم حصول أي استثناء ضمن المقطع **try** وبالتالي لن يتم تنفيذ أي تعليمة **catch**.
- الشكل العام للتعليمة **throw** يكون من الشكل :

```
throw exception;
```

- تقوم التعليمة **throw** بتوليد الاستثناء المحدد بـ **exception**. إذا كان هذا الاستثناء يجب أن يلتقط عندئذ يجب أن يتم تنفيذ **throw** سواء ضمن المقطع **try** أو من قبل أي تابع يتم استدعاؤه ضمن المقطع **try**.
- إذا تم قذف استثناء ليس هناك أي تعليمة **catch** صالحة للتعامل معه، فقد يتم إنهاء البرنامج بطريقة غير متوقعة، إن هذه العملية تؤدي إلى تنفيذ تابع المكتبة القياسية المسمى **terminate()**. افتراضياً يقوم التابع **terminate()** باستدعاء التابع **abort()** لإيقاف تنفيذ البرنامج. (من الممكن تخصيص عمل التابع **terminate()**).
- فيما يلي مثال بسيط يوضح الطريقة التي تعمل بها معالجة الاستثناءات في لغة C++:

```
// exceptionHandling.cpp : main project file.
// A simple exception handling example.
 #include "stdafx.h"
#include <iostream>
using namespace std;
void main()
{      cout << "Start\n";
       try { // start a try block
              cout << "Inside try block\n";
              throw 100; // throw an error
              cout << "This will not execute";    }
       catch (int i) { // catch an error
                     cout << "Caught an exception -- value is: ";
                     cout << i << "\n";       }
cout << "End \n"; system("pause");  } // end main
```

الخرج:

```
Start
Inside try block
Caught an exception -- value is: 100
End
Press any key to continue . . .
```

- بالتدقيق نرى أن هناك مقطع **try** يتضمن ثلاث تعليمات وتعليمة **catch(int i)** تقوم بمعالجة استثناء من النوع الصحيح.
- سيتم تنفيذ اثنتين من التعليمات الثلاث ضمن المقطع **try** : التعليمة **cout** والتعليمة **throw**.
- بمجرد أن يتم قذف الاستثناء يتم نقل التحكم إلى التعليمة **catch** وإنهاء المقطع **try**، أي أن التعليمة **catch** لايتم استدعاؤها وإنما يتم نقل التحكم إليها
- يتم تصفير مكدس البرنامج تلقائياً بما يتناسب مع تحقيق هذا الأمر، وبالتالي فإن التعليمة **cout** التالية للتعليمة **throw** لن تنفذ مطلقاً.
- عادة، يحاول مقطع الشيفرة المضمن في المقطع **catch** أن يعالج خطأ ما بإجراء الفعل المناسب.

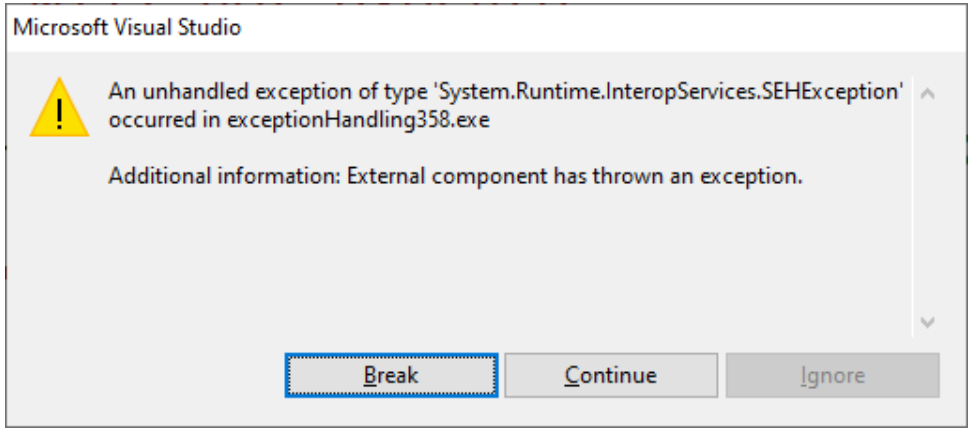
6- 1- المفاهيم الأساسية لمعالجة الاستثناءات

- يتم متابعة التنفيذ للأجزاء التي تلي المقطع catch في حال تمت معالجة الخطأ ضمن المقطع catch، أما في حال لم يتم التمكن من معالجة الخطأ، فإن المقطع catch سيقوم بإنهاء البرنامج باستدعاء التابع exit() أو abort().
- إن نوع الاستثناء يجب أن يتوافق مع النوع المحدد في التعليمة catch.
- إذا تم تغيير النوع في التعليمة catch إلى النوع double فإن الاستثناء لن يلتقط وسيحصل إنهاء طبيعي للبرنامج مع تعليق حسب جيل البيئة، كما يبين الشكل التالي :

```
// This example will not work.
#include "stdafx.h"
#include <iostream>
using namespace std;
void main()
{
cout << "Start\n";
```

```
try { // start a try block
cout << "Inside try block\n";
throw 100; // throw an error
cout << "This will not execute";
}
catch (double i) { // won't work for an int exception
cout<<"Caught an exception -- value is: ";
cout << i << "\n";
}
system("pause");
} // end main
```

سيكون الخرج وفق Visual Studio 10 كما في الحاله السابقة مع رسالة تفيد بما يلي : abnormal program termination



6- 1- المفاهيم الأساسية لمعالجة الاستثناءات

- يمكن أن يتم قذف الاستثناء من خارج المقطع try طالما أنه يقذف من قبل تابع تم استدعاؤه ضمن مقطع try.
- على سبيل المثال، البرنامج التالي يعتبر صالحاً :

```
/* Throwing an exception from a function outside the try block.*/
#include "stdafx.h"
#include <iostream>
using namespace std;
void Xtest(int test)
{     cout << "Inside Xtest, test is: " << test << "\n";
if(test) throw test;}
int main()
{
cout << "Start\n";
```

6- 1- المفاهيم الأساسية لمعالجة الاستثناءات

```
try { // start a try block
      cout << "Inside try block\n";
      Xtest(0);   Xtest(1);   Xtest(2);   }
catch (int i) { // catch an error
      cout << "Caught an exception -- value is: ";
      cout << i << "\n";}     cout << "End \n";
      system("pause");
} // end main
```

سيكون الخرج :

```
Start
Inside try block
Inside Xtest, test is: 0
Inside Xtest, test is: 1
Caught an exception -- value is: 1
End
Press any key to continue . . .
```

6- 1- المفاهيم الأساسية لمعالجة الاستثناءات

- يمكن للمقطع try أن يكون محلياً ضمن تابع ما، عند القيام بذلك فإنه في كل مرة يتم الدخول فيها إلى التابع فإن معالجة الاستثناء الخاص بهذا التابع يتم تصفيرها، المثال التالي يوضح هذه الخاصية :

```
#include "stdafx.h"
#include <iostream>
using namespace std;
// Localize a try/catch to a function.

void Xhandler(int test)
{      try
              {if(test) throw test;     }

       catch(int i)
              {cout << "Caught Exception #: " << i << '\n'; }
}

int main()
{
```

```
    cout << "Start\n";
    Xhandler(1);
    Xhandler(2);
    Xhandler(0);
    Xhandler(3);
cout << "End"<<endl;
system("pause");
} // end main
```

- الخرج كماهو واضح تم قذف ثلاث استثناءات، وبعد كل استثناء يتم إنهاء التابع، وعند استدعاء التابع ثانية يتم تصفير معالجة الاستثناء.

```
Start
Caught Exception #: 1
Caught Exception #: 2
Caught Exception #: 3
End
Press any key to continue . . .
```

6- 1- المفاهيم الأساسية لمعالجة الاستثناءات

من المهم أن نفهم أن الشيفرة المرفقة بالتعليمة catch ستنفذ فقط إذا تمكن التابع من التقاط الاستثناء وإلا فإن التنفيذ سيتجاوز المقطع catch.

مثلاً: في البرنامج التالي لايتم قذف أي استثناء، وبالتالي فإن المقطع catch لن ينفذ.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

int main()
{
cout << "Start\n";

try { // start a try block
        cout << "Inside try block\n";    cout << "Still inside try block\n";
    }
catch (int i) { // catch an error
        cout << "Caught an exception -- value is: ";  cout << i << "\n";
        }
```

```
        cout << "End \n\n";
        system("pause");
        return 0;
}
```

نجد في الخرج: نرى أن المقطع catch تم تجاوزه أثناء تنفيذ البرنامج.

```
Start
Inside try block
Still inside try block
End

Press any key to continue . . .
```

Catching Class Types

التقاط الأصناف

- يمكن للاستثناء أن يكون من أي نوع، بما في ذلك أنماط الأصناف التي تقوم بإنشائها.
-
- إن أغلب الاستثناءات هي من نمط صنف وليس من أنماط مسبقة التعريف. والسبب الأكثر شيوعاً الذي يجعلك ترغب بتعريف نمط صنف هو لإنشاء غرض يصف الخطأ الحاصل. إن هذه المعلومة يمكن أن تستخدم من قبل معالج الاستثناء لمساعدتك على معالجة الخطأ.
- يوضح المثال التالي هذه الخاصية :

```
// Catching class type exceptions.

#include "stdafx.h"
#include <iostream>
#include <cstring>

using namespace std;
```

Catching Class Types

التقاط الأصناف

```
class MyException
{
public:
   char str_what[80];
   int what;
   MyException() { *str_what = 0; what = 0; }
   MyException(char *s, int e) {
   strcpy(str_what, s);
   what = e;
}
};

int main()
{
```

Catching Class Types

التقاط الأصناف

```
int i;
try {
      cout << "Enter a positive number: ";
            cin >> i;
      if(i<0)
      throw MyException("Not Positive", i);
      }
catch (MyException e)
      { // catch an error
      cout << e.str_what << ": ";
      cout << e.what << "\n";
      }
system("pause");
}// end main
```

Catching Class Types

التقاط الأصناف

فيما يلي عينة عن تنفيذ البرنامج :

```
Enter a positive number: -3
Not Positive: -3
Press any key to continue . . .
```

- يطلب البرنامج من المستخدم أن يقوم بإدخال عدد صحيح موجب، فإن قام بإدخال عدد سالب، عندها يتم إنشاء غرض من الصنف **MyException** يصف الخطأ. وبالتالي يغلف الصنف **MyException** المعلومات حول الخطأ، وهذه المعلومات تستخدم بعدئذ من قبل معالج الاستثناء.

- عموماً، سترغب بإنشاء أصناف الاستثناء التي ستغلف المعلومات حول الخطأ لتميكن معالج الاستثناء للاستجابة إليها بفعالية.

Using Multiple Catch Statements

استخدام عدة تعليمات catch

- بشكل عام يمكن أن يكون هناك أكثر من تعليمة **catch** مرتبطة بالكتلة **try**. إلا أنه يجب على كل تعليمة **catch** أن تقوم بالتقاط نوع مختلف من الاستثناءات.
- مثلاً: البرنامج التالي يلتقط استثناءات من النوع **int** و **string** :

```
#include "stdafx.h"
#include <iostream>
using namespace std;
// Different types of exceptions can be caught.
void Xhandler(int test)
{try {if(test) throw test; else throw "Value is zero"; }
catch(int i) {   cout << "Caught Exception #: " << i << '\n';}
catch(const char *str) {cout << "Caught a string: ";
cout << str << '\n';  }
}
```

Using Multiple Catch Statements

استخدام عدة تعليمات catch

```
int main()
{     cout << "Start\n";     Xhandler(1);     Xhandler(2);
      Xhandler(0);           Xhandler(3);     cout << "End"<<endl;
      system("pause");return 0;
}
```

الخرج : كل تعليمة catch تستجيب للنوع الخاص المحدد لها.

```
Start
Caught Exception #: 1
Caught Exception #: 2
Caught a string: Value is zero
Caught Exception #: 3
End
Press any key to continue . . .
```

يتم التحقق من تعابير catch بنفس ترتيب ورودها في البرنامج، ولايتم تنفيذ سوى التعليمة المطابقة وجميع مقاطع catch الأخرى يتم تجاهلها.

2-6- التعامل مع استثناءات من أصناف مشتقة

Handling Derived-Class Exceptions

- يجب توخي الحذر عند ترتيب تعليمات **catch** لدى محاولة التقاط استثناءات من أنواع قد تكون من أصناف أساس أو أصناف مشتقة، لأن النوع المبين في تعليمة **catch** لاستثناء من صنف أساس يتناسب في الوقت ذاته مع الاستثناء من أي صنف مشتق. وبالتالي فإذا أردت التقاط استثناءات من نوع صنف أساس ونوع صنف مشتق يجب وضع الصنف المشتق أولاً في تتالي تعليمات **catch**، فإن لم يتم القيام بذلك فإن تعليمة **catch** من نوع الصنف الأساس ستقوم بالتقاط الاستثناءات الخاصة بالأصناف المشتقة.
- البرنامج التالي يوضح ذلك:

```
// Catching derived classes.
#include "stdafx.h"
#include <iostream>
using namespace std;

class B {
};
```

Handling Derived-Class Exceptions

2-6- التعامل مع استثناءات من أصناف مشتقة

```
class D: public B {
};
int main()
{
      D derived;
      try {
            throw derived;
            }
      catch(B b) { cout << "Caught a base class.\n";    }

      catch(D d) {cout << "This won't execute.\n";      }

system("pause");return 0;
}// end main
```

2-6- التعامل مع استثناءات من أصناف مشتقة

Handling Derived-Class Exceptions

الخرج:

```
Caught a base class.
Press any key to continue . . .
```

- بما أن **derived** هو غرض من مشتق من الصنف الأساس **B** فإنه سيلتقط من قبل أول تعليمة **catch** ولن يتم تنفيذ تعليمة **catch** الثانية مطلقاً.
- تعطي بعض المترجمات رسالة تحذيرية في مثل هذه الحالة، وقد يعطي البعض الآخر رسالة خطأ وفي كلتا الحالتين ما عليك سوى تبديل ترتيب المقطعين **catch** في البرنامج.

- لتحصل على الخرج الصحيح:

```
This won't execute.
Press any key to continue . . .
```

انتهت تمارين الأسبوع العاشر