



محتويات الوحدة

الموضوع	الصفحة
المقدمة	219
تمهيد	219
أهداف الوحدة	220
1. الاستثناء	221
1.1 أساليب معالجة الاستثناءات	221
2. فصول الاستثناءات و كيفية استخدامها.	228
خلاصة	236
المراجع	237

المقدمة

تمهيد

مرحباً بك عزيزي الدارس في هذه الوحدة
يستطيع المبرمج أن يحيط أماكن الشك التي يتوقع أن تحوي أخطاء منطقية في البرامج (debugging) بأوامر خاصة تترقب الأخطاء المحتملة وتقوم بمعالجتها حين وقوعها. في هذا الفصل نتناول أوامر الاستثناء (Exceptions) وكيفية استخدامها في البرامج. في هذه الوحدة ادرجنا تدريبات واسئلة تقويم ذاتي حتى تتمكن من استيعابها وفهمها. هناك نوعان من الأخطاء البرمجية:

- أ- أخطاء تركيبية تتعلق بقواعد اللغة، و هذه عادةً يكتشفها ترجمان اللغة قبل البدء في تشغيل البرنامج.
 - ب- أخطاء منطقية لا يتم اكتشافها إلا بعد البدء في تشغيل البرنامج.
- الأخطاء المنطقية أصعب اكتشافاً من الأخطاء التركيبية اللغوية وقد يكون لها نتائج وخيمة على أداء البرنامج، وربما يتأخر اكتشافها حتى بعد إصدار البرنامج.
- بعض الأخطاء المنطقية يتسبب في
1. تعطيل البرنامج،
 2. يؤدي إلى نتائج خاطئة .

إدراكاً لخطورة هذا النوع من الأخطاء، أصبحت الكثير من لغات البرمجة الحديثة توفر بعض الأساليب التي تساعد في اكتشافها و معالجتها بما يناسب المقام.

الأخطاء المنطقية (التي تحدث أثناء تشغيل البرنامج) يطلق عليها "استثناءات" (Exceptions) في مصطلح العاملين في مجال هندسة البرمجيات.

أهداف الوحدة



بعد فراغك عزيزي الدارس من هذه الوحدة ينبغي أن تكون قادراً على أن:

1. تعرف أنواع الأخطاء البرمجية وتميز الاستثناءات.
2. تستخدم أساليب معالجة الاستثناءات.

1. الاستثناء (exception)

يُوصف بأنه حالة شاذة تحدث أثناء تنفيذ البرنامج.

مثل:

- محاولة القسمة على صفر.
- عدم وجود فراغ كافٍ في الذاكرة لاستيعاب قيمة متغير معين.

1.1 أساليب معالجة الاستثناءات

Exception Handling Techniques

نحاول التنبؤ بهذه الاستثناءات، و معالجتها بالصورة المناسبة.

1. أحياناً يكون من المناسب طباعة رسالة تدل على نوع الاستثناء الذي حدث و الخروج من البرنامج.
2. أحياناً أخرى يكون الأنسب معالجة الاستثناء و الاستمرار في البرنامج، وكأن شيئاً لم يحدث.

C++ توفر ثلاثة عوامل لمعالجة الاستثناءات هي: `try` و `throw` و `catch`. هذه العوامل لابد أن تكون متتالية في البرنامج، لايفصل بينها شيء، ويكون استخدامها عادةً بالصورة التالية:

```
try {  
    // code to be tried  
    throw exception;  
}  
catch (type of exception)  
{  
    // code to be executed in case of  
    exception  
}
```

من الهيكل العام أعلاه يتضح أن :

كلمة try

- تكون متبوعة بفقرة برمجية تحتوي على الأوامر المراد محاولة تنفيذها (منطقة الشك) بالإضافة إلى جملة throw.
- عادةً تحاول جملة try تنفيذ الأوامر المدرجة داخلها، فإذا فشلت استخدمت جملة throw في إثارة الاستثناء.

جملة throw

- تكون متبوعة بالاستثناء (Exception) الذي يكون عادة:
- متغيراً من أي نوع.

- جملة عادية تعطي نوع الخطأ.

- رقماً صحيحاً يدل على الخطأ .

- كيانه متكاملاً يعطي نوع الخطأ .

بعض الطرق التي يمكن أن تستخدم في معالجة الخطأ:

جملة catch التي ينبغي أن تأتي مباشرة بعد جملة try، مهمتها استقبال الاستثناءات التي تثيرها جملة try ومعالجتها.

لاحظ أن جملة catch تستخدم عنصر ممر. الشرط الأساسي في استقبال الاستثناءات المثارة هو أن يتوافق نوع الاستثناء المثار مع نوع العنصر الممر لجملة catch. كلما أثارت جملة try استثناءً، تنظر جملة catch في نوع الاستثناء المثار، فإذا كان متوافقاً مع نوع العنصر الممر، استقبلته وعالجته، وإلا تجاهلته.

مثال (1)

المثال التالي يوضح عملياً كيف تثار الإستثناءات و كيف تستقبل و تعالج. يبدأ البرنامج بتعريف مصفوفة بسيطة ومتغيراً حرفياً OutOfRange يمثل نوع الاستثناء. بعد ذلك يحاول البرنامج تنفيذ أوامر تقوم بتعبئة المصفوفة (داخل جملة try). إذا واجه البرنامج حالة تكون فيها n أكبر من حجم المصفوفة يقوم بإثارة استثناء من النوع

OutOfRangeException. جملة catch تتلقف هذا الإستثناء وتقوم بمعالجته لأنه متوافق مع نوع العنصر الممرر e. المعالجة في هذه الحالة تقتصر على طباعة رسالة تبين نوع الخطأ الذي حدث.

```
// exceptions
#include <iostream>
using namespace std;
int main () {
    int IntArray[10];
    string OutOfRange ="index out of range";
    try
    {
        for (int n=0; n<=10; n++)
        {
            if (n>9) throw OutOfRange; // إثارة الاستثناء
            IntArray[n]=n*n;
        }
    }
    catch (string e)
    {
        cout << "Exception: " << e << endl; // معالجة
        الاستثناء
    }
    cout <<"Note that the program does not
    terminate in this case"<<endl;
    system("pause");
    return 0;
}
```

```
C:\Documents and Settings\affendi\My Documents\1-C++\Book2\Source\ch-8\except-1.exe
Exception: index out of range
Note that the program does not terminate in this case
Press any key to continue . . .
```

لاحظ أن البرنامج يستمر في التنفيذ بعد جملة catch، رغم إثارة الإستثناء. السبب في ذلك أن جملة catch اختارت هذا الأسلوب في معالجة الخطأ ولم تقم بإيقاف البرنامج. أحياناً تستدعي المعالجة إيقاف البرنامج أو تعليقه.

نشاط(1)

أعط بعض الأمثلة للأخطاء التركيبية و المنطقية في برامج C++ ونفذها في جهازك.



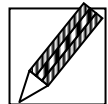
أسئلة تقويم ذاتي

- 1- ما الفرق بين الأخطاء التركيبية و الأخطاء المنطقية؟
- 2- ما الاستثناءات؟ وكيف يتم علاجها في لغة C++
- 3- ما الهيكل العام لجملة try...catch؟



التدريب(1)

- 1- نفذ البرنامج المعطى في مثال(1) من غير استخدام جملة try...catch و انظر ماذا يحدث. استخدم 100 بدلاً من 10 كحد أقصى للمتغير n في جملة for، ثم كرر تنفيذ البرنامج من غير جملة try...catch و انظر ما يحدث.
- 2- اكتب برنامجاً يحسب رواتب العمال في مؤسسة معينة (عدد الساعات مضروباً في أجر الساعة الواحدة) ويثير استثناءً إذا كان الناتج (الراتب) أعلى من 10000.



1.1.1 استخدام جمل catch المتعددة

أحياناً يتطلب المقام إثارة أكثر من استثناء في جملة try واحدة، وبالتالي استخدام أكثر من جملة catch واحدة بعد جملة try. C++ تسمح باستخدام جملة throw أكثر من مرة داخل جملة try لإثارة أكثر من استثناء. المثال التالي يوضح كيف يتم ذلك:

مثال (2)

في هذا المثال جملة try تستخدم جملتي throw، إحداها تثير استثناءً من النوع OutOfRange و الأخرى تثير استثناءً من النوع Overflow. هناك جملتا catch بعد جملة try إحداها لمعالجة الاستثناء الأول (العنصر الممرر متغير ألفبائي) والأخرى لمعالجة الإستثناء Overflow (العنصر الممرر متغير صحيح).

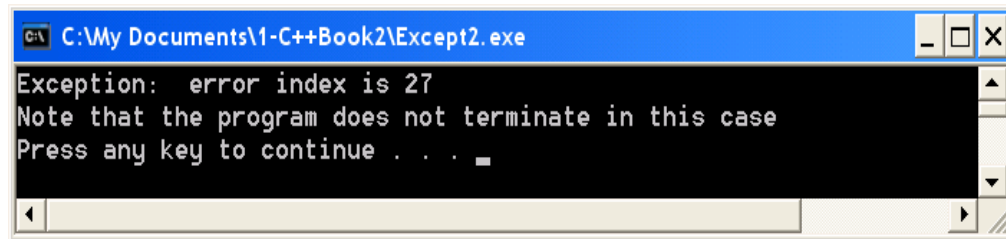
```
// exceptions: multiple catch blocks
#include <iostream>
using namespace std;
int main () {
    int IntArray[10];
    string OutOfRange ="index out of range";
    int Overflow=27;
    int sum;
    try
    {
        for (int n=0; n<=10; n++)
        {
            if (n>9) throw OutOfRange; //إثارة الإستثناء الأول
            {IntArray[n]=n*n;
            if (sum>200) throw Overflow; //إقارة الإستثناء الثاني
                sum=sum+IntArray[n];
            }
        }
    }
    catch (string e) //معالجة الإستثناء الأول
```

```

{
    cout << "Exception: " << e << endl;
}
catch (int e) // معالجة الإستثناء الثاني
{
    cout << "Exception: error index is " << e
<< endl;
}

    cout << "Note that the program does not
terminate in this case" << endl;
    system("pause");
    return 0;
}

```



2.1.1 استخدام جملة catch عامة

أحياناً، يصعب على المبرمج التنبؤ بنوع الخطأ الذي سيحدث، ويهمه فقط أن يعرف أن خطأً من نوع ما قد حدث في منطقة معينة من البرنامج، أو يفضل أن يعالج كل الأخطاء المحتملة باستخدام جملة catch واحدة. C++ تسمح باستخدام جملة catch معممة يمكن استخدامها في مثل هذه الحالات. جملة catch المعممة تستخدم ثلاثة نقاط أفقية متتالية بدلاً من العنصر الممرر، كنايةً عن أنها لا يهتمها نوع الإستثناء.

الشكل العام لجملة catch المعممة هو:

```
try {
```

```

    // code here
}
catch (...) // لاحظ استخدام الثلاث نقاط
{
    cout << "Exception occurred";
}

```

مثال (3)

في هذا المثال نضيف جملة **catch** معممة للبرنامج الذي قدمناه في المثال السابق. جملة **catch** المعممة تستخدم في التعامل مع الاستثناءات غير المعروفة:

```

// exceptions: multiple catch blocks
#include <iostream>
using namespace std;
int main () {
    int IntArray[10];
    string OutOfRange ="index out of range";
    int OverFlow=27;
    int sum;
    try
    {
        for (int n=0; n<=10; n++)
        {
            if (n>9) throw OutOfRange; // إثارة الإستثناء الأول
            {IntArray[n]=n*n;
                if (sum>200) throw OverFlow; // إقارة الإستثناء
الثاني
                sum=sum+IntArray[n];
            }
        }
    }
    catch (string e) // معالجة الإستثناء الأول
    {

```

```

        cout << "Exception: " << e << endl;
    }
    catch (int e) // معالجة الإستثناء الثاني
    {
        cout << "Exception: error index is " << e
        << endl;
    }
    catch (...) // معالجة الإستثناءات غير المعروفة
    {
        cout << "An exception of unknown type
        occurred" << endl;
    }

    cout << "Note that the program does not
    terminate in this case" << endl;
    system("pause");
    return 0;
}

```

2. فصائل الاستثناءات و استخداماتها Exception classes

في الأمثلة السابقة استخدمنا أنواعاً عادية من الفصائل لتعريف الاستثناءات. في التطبيقات العملية الواقعية يفضل استخدام فصائل (classes) كاملة لتعريف الاستثناءات. هذا يساعد كثيراً في معالجة الإستثناءات و إعطاء حلول متكاملة في حالة حدوثها. المثال التالي يوضح كيف يمكن تصميم و إستخدام فصائل كاملة للاستثناءات.

مثال(4)

في هذا المثال نصمم فصيلة استثناء اسمها **DivideByZero** تستخدم في حالة محاولة القسمة على صفر. هذه الفصيلة تستخدم داخل البرنامج في إثارة استثناء عند محاولة القسمة على صفر و تعطي معلومات مفصلة عن حالة البرنامج لحظة حدوث الإستثناء (لا تكتفي بمجرد رسالة خطأ).

برنامج(4)

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

class DivideByZero {
public:
    DivideByZero(int n, int d)
    { num=n; denom=d; message="Divide by
zero"; }
    ~DivideByZero() {}

    int getNumerator() {return num;}
    int getDenominator() {return denom;}
    string getMessage() {return message;}

private:
    int num;
    int denom;
    string message;
};

int main()
{
    int x = 5;
    int y = 0;
    int result;

    try {
```

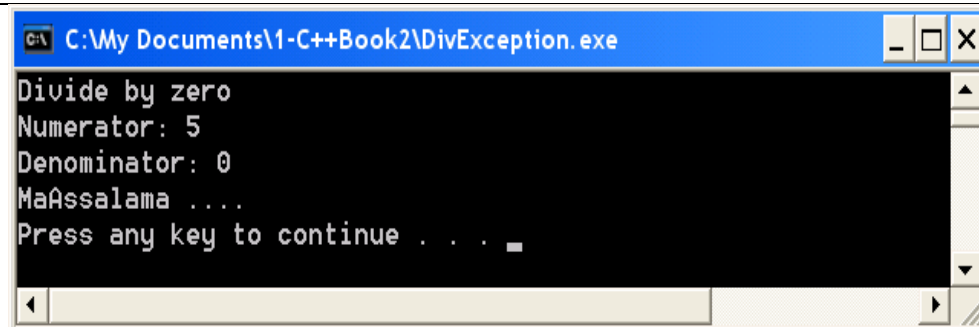
```

        if (y == 0) {
            throw DivideByZero(x, y);
        }

        result = x/y;
    }
    catch (DivideByZero e) {
        cout << e.getMessage() << endl;
        cout << "Numerator: " <<
e.getNumerator() << endl;
        cout << "Denominator: " <<
e.getDenominator() << endl;
    }

    cout << "MaAssalama ...." << endl;
    system("pause");
    return 0;
}

```



```

C:\My Documents\1-C++Book2\DivException.exe
Divide by zero
Numerator: 5
Denominator: 0
MaAssalama ....
Press any key to continue . . .

```

البرنامج أعلاه استخدم الفصيلة **DivideByZero** لمعالجة الاستثناء المتوقع. الشكل العام لهذه الفصيلة هو:

```
class DivideByZero {
public:
    DivideByZero(int n, int d)
    { num=n; denom=d; message="Divide by
zero";}
    ~DivideByZero() {}

    int getNumerator() {return num;}
    int getDenominator() {return denom;}
    string getMessage() {return message;}

private:
    int num;
    int denom;
    string message;
};
```

المفيد في هذه الفصيلة أنها تحتوي على معلومات متكاملة عن حالة البرنامج لحظة حدوث الإستثناء. المعلومات في هذه الحالة تتمثل في قيمة البسط و المقام. هذا يسهل على المبرمج معالجة الخطأ.

لاحظ استخدام الفصيلة **DivideByZero** في جملة **throw**:

throw DivideByZero(x,y);

جملة **throw** في هذه الحالة تشير استثناءً من النوع **DivideByZero**. جملة **catch** تستقبل هذا الإستثناء و تستخلص المعلومات اللازمة من الكيان الممرر.

مثال(5)

في هذا المثال نصمم فصيلة **Stack** تستخدم فصيلتي استثناء: **StackOverflow** و **StackEmpty**. كما سبق أن أوضحنا في الوحدة السابعة أن المرصعة (stack) عبارة عن نوع خاص من القوائم يمكن التعامل معه من جهة واحدة فقط هي أعلى القائمة.

الإضافة و الحذف تتم من أعلى القائمة فقط. عملية الإضافة في هذا المثال تقوم بها الدالة push() وعملية الحذف تقوم بها الدالة pop().

الدالة push() تستخدم فصيلة **StackOverflow** لإثارة استثناء عند محاولة إضافة عنصر إلى مرصعة ممتلئة، بينما تستخدم الدالة push() الفصيلة **StackEmpty** لإثارة استثناء عند محاولة حذف عنصر من مرصعة خالية.

```
#include <iostream>
#include <string>
using namespace std;
class StackOverflow {
public:
    StackOverflow() {}
    ~StackOverflow() {}
    void getMessage()
    {
        cout << "Exception: Stack Full" << endl;
    }
};
class StackEmpty {
public:
    StackEmpty() {}
    ~StackEmpty() {}
    void getMessage()
    {
        cout << "Exception: Stack Empty" <<
endl;
    }
};
// A stack class
class Stack {
public:
    Stack() : index(-1) {}
    ~Stack() {}
```

```

void push(int item)
{
    index++;

    if (index >= 20)
    {
        throw StackOverflow();
    }

    data[index] = item;
}
int pop()
{
    if (index < 0)
    {
        throw StackEmpty();
    }
    int item = data[index];
    index--;
    return item;
}
private:
    int data[20];
    int index;
};

int main()
{
    Stack stk;
    int item1, item2, val;
    cout << "Enter item1: ";
    cin >> item1 ;
    cout << "Enter item2: ";
    cin >> item2;
    try {
        stk.push(item1);

```

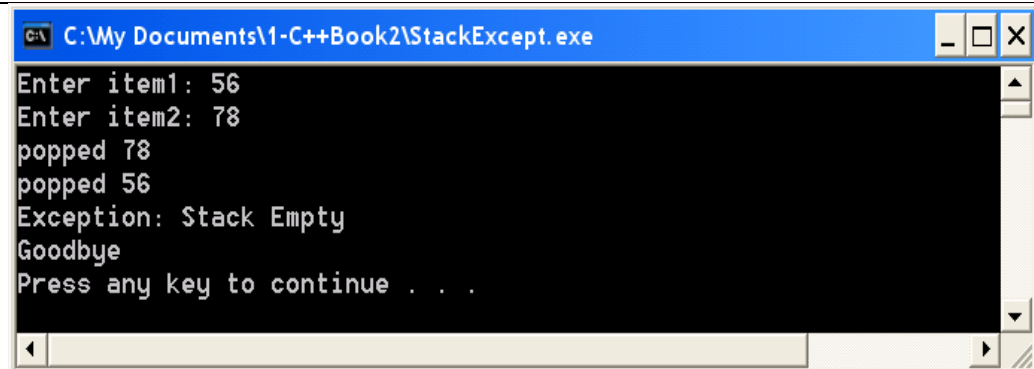
```

        stk.push(item2);

        val = stk.pop();
        cout << "popped " << val << endl;
        val = stk.pop();
        cout << "popped " << val << endl;
        val = stk.pop();
        cout << "popped " << val << endl;
    }
    catch (StackOverflow e) {
        e.getMessage();
    }
    catch (StackEmpty e) {
        e.getMessage();
    }
    catch (...)
    {
        cout << "An exception of unknown type
occurred" << endl;
    }

    cout << "Goodbye" << endl;
    system("pause");
    return 0;
}

```



```

C:\My Documents\1-C++Book2\StackExcept.exe
Enter item1: 56
Enter item2: 78
popped 78
popped 56
Exception: Stack Empty
Goodbye
Press any key to continue . . .

```

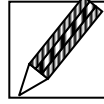
لاحظ أننا استخدمنا ثلاثة جمل **catch** في البرنامج أعلاه:

واحدة : لمعالجة الخطأ الناجم من محاولة الإضافة إلى مرصعة مملوءة.

الثانية: لمعالجة الخطأ الناجم من محاولة الحذف من مرصعة فارغة.

الثالثة: عامة للإخبار بحدوث خطأ غير معروف السبب.

تدريب (2)



1- اكتب برنامجاً يحسب متوسط الدرجات التي حصل عليها الطلاب في مادة معينة. استخدم أوامر الاستثناء في معالجة الحالات التي تكون فيها الدرجات المعطاة أكثر من 100 أو أقل من صفر.

2- أعد كتابة البرنامج أعلاه باستخدام فصيلة استثناء اسمها `GradeAverage`. الفصيلة `GradeAverage` يمكن استخدامها في إثارة استثناء في حالة أن الدرجة أكبر من 100 أو أقل من صفر مع الإستمرار في حساب المتوسط. في النهاية تعطي الفصيلة معلومات عن عدد الدرجات التي استخدمت في حساب المتوسط، مع قائمة بالدرجات التي لم تستخدم في حساب المتوسط (الدرجات التي تكون أكبر من مائة أو أقل من صفر).

3- اكتب فصيلة `SafeArray` تمنع التخزين خارج نطاق المصفوفة. استخدم أوامر الاستثناء في تصميم هذه الفصيلة.

4- بعض دوال مكتبة `C++` القياسية تثير استثناءات معرفة سلفاً في فصول استثناء جاهزة. هذه الفصول مبنية على فصيلة استثناء قياسية: `std::exception`. حاول أن تكتشف الهيكل العام لفصول استثناء الدوال الجاهزة و كيف يمكن استخدامها في البرامج العادية.

الخلاصة

- الأخطاء المنطقية التي تحدث بعد البدء في تشغيل البرنامج تسبب مشاكل كبيرة للمطورين وتطوّل زمن التطوير، وبالتالي تزيد تكلفة تطوير البرامج.
- يمكن تقليل مخاطر الأخطاء المنطقية باستخدام أوامر الإستثناء في لغات البرمجة.
- الأخطاء المنطقية تسمى "الإستثناءات" (exceptions) في عرف مهندسي البرمجيات.
- سي++ توفر ثلاثة أوامر لمعالجة الإستثناءات هي: try catch throw.
- الأمثلة التي قدمناها تشرح كيف يمكن استخدام هذه الأوامر.
- يمكن استخدام فصول متكاملة لمعالجة الإستثناءات. الفائدة التي نجنوها في هذه الحالة هي إعطاء المطور معلومات متكاملة عن حالة البرنامج لحظة حدوث الإستثناء، بالإضافة إلى طرق لمعالجة الإستثناء.

المصادر والمراجع

- 1- Deictl, H. & Deitel, P., **C++ How to Program**. Third Edition, Prentice Hall: 2001.
- 2- Savitch, W. **Problem Solving With C++: The Object of Programming**. Third Edition, Addison Wesley: 1999.
- 3- Friedman, F. and Koffman, E. **Problem Solving, Abstraction, and Design Using C++** (2 nd Edition). Addison Wesley: 1996.
- 4- Lippman, S. Iajoie, J. **C++ Primer**, (3rd Edition). Addison Wesley: 1998.
- 5- Horstmann, C. **Computing Concepts With C++ Essentials**. John Wiley: 1997.
- 6- Stroustrup, B. **The C++ Programming Language**. (3rd edition), Addison Wesley: 1990.
- 7- Ellis, M. and Bjarnic Stroustrup, **The Annotated Reference Manual**. Addison Wesley: 1990.
- 8- Tcale, S. **C++ IOS treams Handbook**. Addison Wesley: 1993.
- 9- Meyers, S. **Effective C++: 50 Specific Ways to Improve Your Programs and Designs** (2 nd edition). Addison Wesley: 1996.
- 10- Meyers, S. **More Effective C++: 35 New Ways to Improve Your Programs and Designs**. Addison Wesley. 1996.
- 11- Schildt, H. **C++ From the Ground up**. Mc Graw Hill. 2003.
- 12- Savitch, W. **Absolute C++**. Addison Wesley: 2002.
- 13- Cogswell, J. **C++ all – in – one Desk Refrence for Dummies**. 2002.