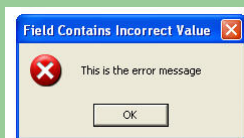


חריגות (exceptions)

אוהד ברזילי
אוניברסיטת תל אביב



טיפול בשגיאות

- חלופות "ידיניות":
- סיום התוכנית
- החזרת ערך המייצג שגיאה
- החזרת ערך חוקי, אך יצירת מצב לא חוקי
- קריאה לפונקציה המטפלת בשגיאות



הצלחה וכשלון



- **הצלחה** – קריאה לפונקציה מצליחה אם ריצת הפונקציה מסתיימת במצב המקיים את החוזה של הפונקציה
- **כשלון** - קריאה לפונקציה נכשלת אם ריצת הפונקציה מסתיימת במצב שאינו מקיים את החוזה של הפונקציה
- **חריגה (exception)** – היא ארוע זמן ריצה, אשר גורם לפונקציה להיכשל

3

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

חריגות

- חריגה מתרחשת במהלך ביצוע פונקציה f אם קורים אחד הבאים:
- מעקב (dereferencing) אחרי מצביע שאינו מוקצה
 - ביצוע פקודה מסוימת זורק (throw) חריגה בצורה מפורשת
 - פונקציה שנקראה ע"י f נכשלה
- הערה: לא כל פעולה שאינה חוקית ב C++ יוצרת חריגה (למשל: נסיון לשחרר זכרון על המחסנית)

4

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

חריגות וחוזים

- כלים אשר אוכפים טענות וחוזים ב C++ בדר"כ עושים זאת ע"י זריקת חריגה עבור ההפרות הבאות:
 - הפרת תנאי קדם בכניסה לפונקציה
 - הפרת תנאי בתר ביציאה מפונקציה
 - הפרת שמורה של מחלקה בכניסה וביציאה של כל פונקציה מיוצאת של אותה מחלקה
 - שמורת הלולאה אינה מתקיימת בכניסה למחזור הראשון או בסופו של כל מחזור
 - משתנה הלולאה (loop variant) לא קטן מהמחזור הקודם
 - ביטוי assert נכשל

5

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

חריגה וכשלון

- קריאה לפונקציה נכשלת אם קרתה חריגה במהלך ריצת הפונקציה והפונקציה לא התאוששה מהחריגה
- כשלון של פונקציה יוצרת חריגה אצל מי שקרא לפונקציה
- מילים שמורות:
 - try : בלוק 'מועד לפורענות' (שעלול ליצר חריגות)
 - catch : בלוק התאוששות מה'פורענות'
 - throw : ייצור של חריגה

6

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

Div

```
/** returns a/b
 * @pre: b>0 , "Div handles only positive b's"
 */
int Div(int a, int b)
{
    if (b == 0)
        throw "Dividing by Zero";
    else if (b < 0)
        throw 0;
    else
        return a/b;
}
```

7

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

```
int main()
{
    try
    {
        int r1 = Div(3,4);
        int r2 = Div(4,0);
        int r3 = Div(2,-5);
    }
    catch (char* msg)
    {
        cout<<"An error occurred!!!!!"<<endl;
        cout<<msg<<endl;
    }
    catch (int num)
    {
        cout<<"An error occurred!!!!!"<<endl;
        cout<<"the number "<<num<<"was caught."<<endl;
    }
    catch (...)
    {
        cout<<"An unknown error occurred!!!!!"<<endl;
    }
}
```

8

ניהול חריגות מתקדם

- ברור סוג החריגה שקרתה
- ניתן לספק טיפול שונה לכל אחת מהחריגות
- ניתן להתעלם מחריגות מסוימות
- ניתן ליזום חריגות בצורה מפורשת כחלק מתזרים התוכנית
- ניתן להשתמש בטיפוסי מחלקה סטנדרטים ובכאלה שהוגדרו ע"י המשתמש בתור טיפוסי העצם הנזרק

9

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

דוגמא: חריגה מתחום

```
struct Range_error {  
    int i;  
    Range_error(int ii) { i = ii; } // constructor  
};  
  
char to_char(int i)  
{  
    if (i < numeric_limits<char>::min() ||  
        numeric_limits<char>::max() < i )  
        throw Range_Error(i);  
    return i;  
}
```

10

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

הערכות לחריגה מתחום

```
void g(int i)
{
    try {
        char c = to_char(i);
        // ...
    }
    catch(Range_error) {
        cerr << "oops\n";
    }
}
```



11

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

שימוש בעצם שנזרק

```
void h(int i)
{
    try {
        char c = to_char(i);
        // ...
    }
    catch(Range_error x) {
        cerr << "oops : to _char(" << x.i << ")\n ";
    }
}
```

12

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

משפחות של עצמים נזרקים

- לעיתים די בהגדרת מחלקה ריקה כדי לציין את סוג השגיאה
- ניתן להגדיר 'משפחות' של מחלקות הנפרדות זו מזו רק בשמן (בהמשך נוכל לממש זאת בקלות ע"י ירושה)



13

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

```
class Overflow {};  
class Underflow {};  
class Zerodivide {};  
  
void f()  
{  
    try {  
        // ...  
    }  
    catch (Overflow) {  
        // handle Overflow or anything derived from  
        // Overflow  
    }  
    catch (Underflow) {  
        // handle Overflow or anything derived from  
        // Underflow  
    }  
    catch (Zerodivide ) {  
        // handle Overflow or anything derived from  
        // Zerodivide  
    }  
}
```

זריקה חוזרת

- כאשר לבלוק ה catch (exception handler) אין מספיק מידע כדי לטפל בבעיה שנוצרה, הוא עשוי לזרוק אותה הלאה
- הדבר נעשה ע"י שימוש במילה throw ללא שם עצם אחריה



תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

15

```
void h()
{
    try {
        // code that might throw Math errors
    }
    catch (Matherr) {
        if (can_handle_it_completely)
        {
            // handle the Matherr
            return ;
        }
        else
        {
            // do what can be done here
            throw; // rethrow the exception
        }
    }
}
```

16

עקרון הפשטות

- מטרתן של הפעולות המתבצעות בבלוק ה- `catch` היא להחזיר את העצם (הספק) למצב יציב:
 - שחרור משאבים (כגון זכרון) אם הוקצו טרם החריגה
 - קיום שמורת המחלקה
 - נסיון חוזר או ויתור במקרה הצורך
- על כל הפעולות בבלוק להיות ממוקדות בהשגת מטרת אלו

17

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

שחרור משאבים

```
void use_file(const char *fn)
{
    FILE* f = fopen(fn , "w");
    // use f
    fclose(f);
}
```

- מה עלול להשתבש?

18

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

שחרור משאבים

```
void use_file(const char *fn)
{
    FILE *f = fopen(fn, "r");
    try {
        // use f
    }
    catch (...)
    {
        fclose(f);
        throw ;
    }
    fclose(f);
}
```



19

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

שחרור משאבים במקרה הכללי

```
void acquire()
{
    // acquire resource 1
    // ...
    // acquire resource n

    // use resources

    // release resource n
    // ...
    // release resource 1
}
```



20

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

שחרור משאבים במקרה הכללי

- נעטוף את המשאבים המוקצים בעצמים אשר ינהלו את הקצאת ושחרור המשאבים בבנאי ובמפרק
- השיטה מודגמת עבור קבצים אולם היא זהה גם עבור נעילות, הקצאות זכרון, קווי תקשורת ועוד...

21

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

```
class File_ptr {  
    FILE *p;  
public:  
    File_ptr(const char *n , const char *a)  
    { p = fopen(n,a); }  
  
    File_ptr(FILE *pp) { p = pp;}  
  
    ~File_ptr() { fclose(p);}  
  
    operator FILE *() { return p ;}  
};
```

22

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

ניהול משאב ע"י עצם עוטף

```
void use_file(const char *fn)
{
    File_ptr f(fn, "r");
    // use f
}
```

- אם במהלך השימוש בקובץ נזרקה חריגה, בעת הריסת העצם המקומי f ישתחרר המשאב המוקצה

23

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

ריבוי משאבים

- שימוש בשיטת העצם העוטף אינו מסבך את המימוש גם עבור ריבוי משאבים

```
class X {
    File_ptr aa;
    Lock_ptr bb;
public:
    X(const char *x, const char *y)
        : aa(x, "rw"), // acquire 'x'
          bb(y)         // acquire 'y'
    {}
    // ...
};
```

24

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

חריגות בנייה



- מה קורה אם init זורק exception ?

```
class Y {
    int *p;
    void init();
public:
    Y(int s) { p = new int[s]; init(); }
    ~Y() { delete [] p ; }
    // ...
};
```

25

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

וקטור בא לעזרה



```
class Z {
    vector<int> p;
    void init();
public:
    Z(int s) : p(s) { init(); }
    // ...
};
```

26

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

auto_ptr

- טכניקת שחרור המשאבים מומשה בספרייה הסטנדרטית (std) ע"י `auto_ptr<T>`
- זהו מצביע כללי המשחרר את הזכרון המוצבע על ידו בעת היציאה מהתחום שבו הוגדר (בין אם ע"י סיום תקין או עקב זריקה של חריגה)

27

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב



עוד על חריגות בנייה

```
class Vector {
public:
    class Size {};
    enum {max = 32000};
    Vector(int sz)
    {
        if (sz<0 || max<sz) throw Size();
        // ...
    }
    // ...
};
```

28

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב



עוד על חריגות בנייה

```
Vector *f(int i)
{
    try {
        Vector *p = new Vector(i);
        // ...
        return p;
    }
    catch(Vector::Size) {
        // deal with size error
    }
}
```

29

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

```
class X {
    Vector v;
    // ...
public:
    X(int);
    // ...
};

X::X(int s)
try
    :v(s) // initialize v by s
    {
        // ...
    }
catch(Vector::Size) { // exceptions thrown for
                      // v are caught here
    // ...
}
```

וכאשר Vector הוא
אחד השדות

30

חריגות בהעתקה ובפירוק

- יש להימנע מיצירת חריגות בתוך בנאי העתקה ובתוך אופרטור ההשמה
 - שני מקרים אלו נקראים בצורה מרומזת אשר קשה מאוד לעטוף בבלוק try
- יש להימנע מיצירת חריגות בתוך מפרק
 - במקרה זה המפרק עשוי להיקרא במסגרת טיפול בחריגה קודמת

31

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

הצהרה על חריגות

- פונקציות ומתודות יכולות להצהיר על מגוון סוגי החריגות שהן זורקות
- לדוגמא:

```
void f(int a) throw(x2 , x3);
```

f רשאית לזרוק רק חריגות מטיפוס x2 או x3

```
int g(); // can throw any exception
```

```
int h() throw (); // no exception thrown
```

32

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

חריגה שאינה שגיאה (אלא מבנה בקרה נוסף)

```

void f(Queue<X>& q)
{
    try {
        for (;;) {
            X m = q.get(); // throws 'Empty' if queue is empty
            // ...
        }
    }

    catch (Queue<X>::Empty) {
        return;
    }
}

```

33

תכנות מונחה עצמים בשפת C++
אוניברסיטת תל אביב

חריגות סטנדרטיות

Standard Exceptions (thrown by the language)			
Name	Thrown by	Reference	Header
<i>bad_alloc</i>	<i>new</i>	§6.2.6.2, §19.4.5	<new>
<i>bad_cast</i>	<i>dynamic_cast</i>	§15.4.1.1	<typeinfo>
<i>bad_typeid</i>	<i>typeid</i>	§15.4.4	<typeinfo>
<i>bad_exception</i>	<i>exception specification</i>	§14.6.3	<exception>

Standard Exceptions (thrown by the standard library)			
Name	Thrown by	Reference	Header
<i>out_of_range</i>	<i>at()</i>	§3.7.2, §16.3.3, §20.3.3	<stdexcept>
	<i>bitset<>::operator[]()</i>	§17.5.3	<stdexcept>
<i>invalid_argument</i>	<i>bitset constructor</i>	§17.5.3.1	<stdexcept>
<i>overflow_error</i>	<i>bitset<>::to_ulong()</i>	§17.5.3.3	<stdexcept>
<i>ios_base::failure</i>	<i>ios_base::clear()</i>	§21.3.6	<ios>

34