Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Αντικειμενοστρεφής Προγραμματισμός (C++)

Ενότητα 8: Χώροι ονομάτων-εξαιρέσεις

Καθηγήτρια Κλειώ Σγουροπούλου

Χώροι ονομάτων (namespaces) 1/4

- Επιτρέπουν την ομαδοποίηση ενός συνόλου καθολικών κλάσεων, αντικειμένων και/ή συναρτήσεων με βάση ένα συγκεκριμένο όνομα. Επιτρέπουν δηλ. τον διαχωρισμό του χώρου καθολικής εμβέλειας σε υπο-χώρους εμβέλειας
  - **namespace** ***identifier*** **{**

    namespace-body

    };

```
namespace general
{
  int a, b;
}
```

Προσπέλαση μεταβλητών a,b

```
general::a
general::b
```

Χώροι ονομάτων (namespaces) 2/4

- Οι χώροι ονομάτων είναι ιδιαίτερα χρήσιμοι στις περιπτώσεις όπου είναι πιθανό ένα καθολικό αντικείμενο/συνάρτηση να έχει το ίδιο όνομα με κάποιο άλλο, προκαλώντας έτσι σφάλμα επανορισμού.

```
#include <iostream.h>

namespace first {
  int var = 5;}

namespace second {
  double var = 3.1416;}

int main () {
  cout << first::var << endl;
  cout << second::var << endl;
  return 0;
}
```

Χώροι ονομάτων (namespaces) 3/4

- Η οδηγία **using namespace** εξυπηρετεί στο συσχετισμό του παρόντος επιπέδου εμφώλευσης με ένα συγκεκριμένο χώρο ονομάτων, ώστε τα αντικείμενα και οι συναρτήσεις του εν λόγω χώρου να είναι άμεσα προσπελάσιμες, σαν αυτές να είχαν οριστεί ως καθολικές
  - **using namespace** *identifier;*

```
namespace first {int var = 5;}

namespace second {double var = 3.1416;}

int main () {
  using namespace second;
  cout << var << endl;
  cout << (var*2) << endl;
  return 0;}
```

Χώροι ονομάτων (namespaces) 4/4

- Η πρόταση `using namespace` έχει ισχύ μόνο στο μπλοκ μέσα στο οποίο ορίζεται ή σε όλο το πρόγραμμα αν δηλώνεται καθολικά

```
namespace first {int var = 5;}

namespace second {double var = 3.1416;}

int main () {
  {
    using namespace first;
    cout << var << endl;
  }
  {
    using namespace second;
    cout << var << endl;
  }
  return 0;
}
```

Namespace `std`

- Το καλύτερο παράδειγμα χώρου ονομάτων είναι η ίδια η βιβλιοθήκη της C++. Όπως ορίζεται στο πρότυπο ANSI C++, όλες οι κλάσεις, τα αντικείμενα και οι συναρτήσεις της βιβλιοθήκης ορίζονται στον χώρο ονομάτων **`std`**.
- Με το πρότυπο ANSI η βιβλιοθήκη της C++ επανασχεδιάστηκε ώστε να χρησιμοποιεί τα χαρακτηριστικά των `templates`

```
// ANSI-C++ compliant hello world
#include <iostream>
//using namespace std;

int main () {
  std::cout << "Hello world in ANSI-C++\n";
  return 0;
}
```

Χειρισμός εξαιρέσεων 1/5

- Κατά τη διάρκεια ανάπτυξης ενός προγράμματος, μπορεί να υπάρξουν περιπτώσεις, όπου δεν έχουμε τη βεβαιότητα ότι το συγκεκριμένο κομμάτι κώδικα θα δουλέψει σωστά, είτε επειδή μπορεί να προσπαθήσει να προσπελάσει πόρους που δεν υπάρχουν, είτε επειδή βγαίνει εκτός ορίων, ...
- Αυτές οι περιπτώσεις ανωμαλίας εντάσσονται σε αυτό που ονομάζουμε **εξαιρέσεις**, για το χειρισμό των οποίων η C++ χρησιμοποιεί τους τελεστές `try`, `throw`, `catch`
  -
```
try { // code to be tried
      throw exception;}
catch (type  exception) {
// code to be executed in case of exception
}
```

Χειρισμός εξαιρέσεων 2/5

- Ο κώδικας στο μπλοκ **try** εκτελείται κανονικά
- Αν κάτι συμβεί ο κώδικας χρησιμοποιεί τη λέξη **throw** και μια παράμετρο για να προκαλέσει εξαίρεση
- Αν έχει συμβεί μια εξαίρεση, το μπλοκ **catch** εκτελείται λαμβάνοντας ως παράμετρο αυτή που περάστηκε από το **throw**

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

Χειρισμός εξαιρέσεων 3/5

```
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 (char * str)
  {cout << "Exception: " << str << endl;}
  return 0;
}
```

Χειρισμός εξαιρέσεων 4/5

- Μπορούμε να ορίζουμε ένα μοναδικό **catch** μπλοκ ανεξάρτητα από τον τύπο που χρησιμοποιήθηκε στο **throw**
  - ```
    try {
      // code to be tried
    }
    catch (…) {
      cout << "Exception occurred";
    }
    ```
    ```

Χειρισμός εξαιρέσεων 5/5

- Μπορούμε να ορίζουμε εμφωλευμένα `try-catch` μπλοκς
  -
```
try {
      try {
      // code here
     }
     catch (int n) {
       throw;
     }
}
catch (...) {
  cout << "Exception occurred";
}
```

Τυπικές εξαιρέσεις 1/2

- Πολλές συναρτήσεις της βιβλιοθήκης της C++ προκαλούν εξαιρέσεις οι οποίες μπορούν να 'πιαστούν' αν τις συμπεριλάβουμε σε ένα **`try`** μπλοκ. Κάθε τύπος των εξαιρέσεων αυτών προέρχεται από την κλάση **`exception`**

```
exception
 ├─ bad_alloc          (thrown by new)
 ├─ bad_cast           (thrown by dynamic_cast when fails with a referenced type)
 ├─ bad_exception      (thrown when an exception doesn't match any catch)
 ├─ bad_typeid         (thrown by typeid)
 ├─ logic_error
 │   ├─ domain_error
 │   ├─ invalid_argument
 │   ├─ length_error
 │   └─ out_of_range
 ├─ runtime_error
 │   ├─ overflow_error
 │   ├─ range_error
 │   └─ underflow_error
 └─ ios_base::failure  (thrown by ios::clear)
```

Τυπικές εξαιρέσεις 2/2

Exception: Attempted typeid of NULL pointer

```
#include <iostream.h>
#include <exception>
#include <typeinfo>

class A {virtual f() {}; };

int main () {
  try {
    A * a = NULL;
    typeid (*a);
  }
  catch (std::exception& e)
  {
    cout << "Exception: " << e.what();
  }
  return 0;
}
```

Εξαιρέσεις: παράδειγμα 1/3

```
// Fig. 16.1: DivideByZeroException.h
// Class DivideByZeroException definition.
#include <stdexcept> // stdexcept header file contains runtime_error
using std::runtime_error; // standard C++ library class runtime_error

// DivideByZeroException objects should be thrown by functions
// upon detecting division-by-zero exceptions
class DivideByZeroException : public runtime_error
{
public:
   // constructor specifies default error message
   DivideByZeroException::DivideByZeroException()
      : runtime_error( "attempted to divide by zero" ) {}
}; // end class DivideByZeroException
```

Εξαιρέσεις: παράδειγμα 2/3

```
1  // Fig. 16.2: Fig16_02.cpp
2  // A simple exception-handling example that checks for
3  // divide-by-zero exceptions.
4  #include <iostream>
5  using std::cin;
6  using std::cout;
7  using std::endl;
8
9  #include "DivideByZeroException.h" // DivideByZeroException class
10
11 // perform division and throw DivideByZeroException object if
12 // divide-by-zero exception occurs
13 double quotient( int numerator, int denominator )
14 {
15    // throw DivideByZeroException if trying to divide by zero
16    if ( denominator == 0 )
17       throw DivideByZeroException(); // terminate function
18
19    // return division result
20    return static_cast< double >( numerator ) / denominator;
21 } // end function quotient
22
```

Εξαιρέσεις: παράδειγμα 3/3

```
23  int main()
24  {
25     int number1; // user-specified numerator
26     int number2; // user-specified denominator
27     double result; // result of division
28
29     cout << "Enter two integers (end-of-file to end): ";
30
31     // enable user to enter two integers to divide
32     while ( cin >> number1 >> number2 )
33     {
34        // try block contains code that might throw exception
35        // and code that should not execute if an exception occurs
36        try
37        {
38           result = quotient( number1, number2 );
39           cout << "The quotient is: " << result << endl;
40        } // end try
41
42        // exception handler handles a divide-by-zero exception
43        catch ( DivideByZeroException &divideByZeroException )
44        {
45           cout << "Exception occurred: "
46              << divi
47        } // end catch
48
49        cout << "\nEnt
50     } // end while
51
52     cout << endl;
53     return 0; // ter
54  } // end main
```

```
Enter two integers (end-of-file to end): 100 7
The quotient is: 14.2857
Enter two integers (end-of-file to end): 100 0
Exception occurred: attempted to divide by zero
Enter two integers (end-of-file to end): ^Z
```

ΤΕΛΟΣ ΕΝΟΤΗΤΑΣ

Σημειώματα

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Επεξήγηση όρων χρήσης έργων τρίτων

©	Δεν επιτρέπεται η επαναχρησιμοποίηση του έργου, παρά μόνο εάν ζητηθεί εκ νέου άδεια από το δημιουργό.
διαθέσιμο με άδεια CC-BY	Επιτρέπεται η επαναχρησιμοποίηση του έργου και η δημιουργία παραγώγων αυτού με απλή αναφορά του δημιουργού.
διαθέσιμο με άδεια CC-BY-SA	Επιτρέπεται η επαναχρησιμοποίηση του έργου με αναφορά του δημιουργού, και διάθεση του έργου ή του παράγωγου αυτού με την ίδια άδεια.
διαθέσιμο με άδεια CC-BY-ND	Επιτρέπεται η επαναχρησιμοποίηση του έργου με αναφορά του δημιουργού. Δεν επιτρέπεται η δημιουργία παραγώγων του έργου.
διαθέσιμο με άδεια CC-BY-NC	Επιτρέπεται η επαναχρησιμοποίηση του έργου με αναφορά του δημιουργού. Δεν επιτρέπεται η εμπορική χρήση του έργου.
διαθέσιμο με άδεια CC-BY-NC-SA	Επιτρέπεται η επαναχρησιμοποίηση του έργου με αναφορά του δημιουργού και διάθεση του έργου ή του παράγωγου αυτού με την ίδια άδεια. Δεν επιτρέπεται η εμπορική χρήση του έργου.
διαθέσιμο με άδεια CC-BY-NC-ND	Επιτρέπεται η επαναχρησιμοποίηση του έργου με αναφορά του δημιουργού. Δεν επιτρέπεται η εμπορική χρήση του έργου και η δημιουργία παραγώγων του.
διαθέσιμο με άδεια CC0 Public Domain	Επιτρέπεται η επαναχρησιμοποίηση του έργου, η δημιουργία παραγώγων αυτού και η εμπορική του χρήση, χωρίς αναφορά του δημιουργού.
διαθέσιμο ως κοινό κτήμα	Επιτρέπεται η επαναχρησιμοποίηση του έργου, η δημιουργία παραγώγων αυτού και η εμπορική του χρήση, χωρίς αναφορά του δημιουργού.
χωρίς σήμανση	Συνήθως δεν επιτρέπεται η επαναχρησιμοποίηση του έργου.

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.