ΗΥ352 : ΕΡΓΑΣΤΗΡΙΟ ΤΕΧΝΟΛΟΓΙΑΣ ΛΟΓΙΣΜΙΚΟΥ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

4ο ΦΡΟΝΤΙΣΤΗΡΙΟ

ΔΙΔΑΣΚΟΝΤΑΣ
Αντώνιος Σαββίδης

Περιεχόμενα

- Namespaces
- Input / Output στη C++
- Exceptions

Namespaces (1/4)

- Τα *namespaces* επιτρέπουν την ομαδοποίηση οντοτήτων όπως κλάσεις, αντικείμενα και συναρτήσεις κάτω από ένα ενιαίο όνομα
  - Χωρίζουμε ένα scope σε "sub-scopes", το κάθε ένα με το δικό του όνομα
  - Δήλωση ενός namespace επιτρέπεται μόνο σε global scope ή απευθείας μέσα σε άλλο namespace
- ***namespace identifier { /*declarations*/ }***
  - Στα declarations μπορούμε να έχουμε κλάσεις, αντικείμενα συναρτήσεις, άλλα namespaces, κλπ
  - π.χ. *namespace foo { int a, b; void bar(void) {} class X {}; }*

Namespaces (2/4)

- Μπορούμε να αποκτήσουμε πρόσβαση στις οντότητες ενός namespace με το scope resolution operator **::**
  - *namespace foo { int a, b; class X {}; void bar(X *x) {} }*
  - int x = foo::a; foo::bar(new foo::X());
- Μπορούμε ακόμα να χρησιμοποιήσουμε ένα namespace, κάνοντας μερικές ή όλες τις οντότητες του διαθέσιμες στο τρέχον scope χρησιμοποιώντας το keyword ***using***
  - using namespace foo; bar(new X());
  - using foo::X; X x;
- Μπορούμε να δημιουργήσουμε aliases για namespaces
  - namespace outer { namespace inner { int var; } }
  - namespace n = outer::inner; **//n::var same as outer::inner::var**

Namespaces (3/4)

- Παραδείγματα

```
#include <stdio.h>
namespace A { char c = ‘a’; }
namespace B { char c = ‘b’; }
int main() {
   {   using namespace A;
       printf(“%c”, c); }  // prints a
   using B::c;
   printf(“%c”, c);        // prints b
}
```

```
#include <stdio.h>
namespace A { char c = ‘a’; }
namespace B { char c = ‘b’; }
int main() {
    char c = ‘c’;
    using namespace A;
    using namespace B;
    printf(“%c”, c);    // prints c
}
```

```
#include <stdio.h>
namespace A { char c = ‘a’; }
namespace B { char c = ‘b’; }
char c = ‘c’;
int main() {
    using namespace A;
    using namespace B;
    printf(“%c”, c);  // error, ambiguous
}                     // access to c
```

```
#include <stdio.h>
namespace A { char c = ‘a’; }
namespace B { char c = ‘b’; }
int main() {
    char c = ‘c’;
    using namespace A;
    using B::c; // error, c alredy declared
    printf(“%c”, c);
}
```

Namespaces (4/4)

- Μπορούμε να «ανοίξουμε» και να κλείσουμε ένα namespace όσες φορές και σε όσα αρχεία θέλουμε
- Σε κάθε σημείο του κώδικα βλέπουμε από το namespace μόνο ότι έχει δηλωθεί μέχρι εκεί
- Αυτό το συναντάμε συχνά με διάφορα header files που περιέχουν οντότητες της standard βιβλιοθήκης της C++ μέσα στο namespace ***std***

```
// File X.h
namespace util {
    class X {};
}
```

```
// File Y.h
namespace util {
    class Y {};
}
```

```
// File main.cpp
#include "X.h"
util::X x;
util::Y y;  //error, no Y in util
#include "Y.h"
util::Y y;  //ok, now found
```

Input / Output στη C++ - Streams

- Τα input και output operations στη C++ γίνεται με ***streams***
  - Stream ονομάζουμε ένα αντικείμενο που κρατάει και μορφοποιεί άλλα αντικείμενα.
  - Στη C++ έχουμε διαφορετικά είδη από input streams & output streams.
    - ifstreams, ofstreams (για αρχεία)
    - istrstreams, ostrstreams (για char *)
    - istringstreams, ostringstreams (για std:string)
  - Όλα έχουν το ίδιο interface

Input / Output στη C++ - iostream (1/3)

- Η βασική βιβλιοθήκη της C++ για input / output operations είναι η βιβλιοθήκη ***iostream***
- Περιλαμβάνει και ορίζει τα αντικείμενα ***cout*** και ***cin*** (streams) με το interface των οποίων εκτελούμε τις βασικές λειτουργίες για είσοδο και έξοδο
- Τα cin και cout είναι αντίστοιχα με τα stdin και stdout της C. Δηλαδή σκεφτόμαστε το cout σαν το console output και το cin σαν το console user input

Input / Output στη C++ - iostream (2/3)

- Για τα input και output operations χρησιμοποιούμε τους overloaded τελεστές **>>** και **<<**
- Είναι overloaded για όλους τους βασικούς τύπους
- Μπορούμε να τους κάνουμε overload και για οποιοδήποτε user defined τύπο

```
#include <iostream>

int main(){
   int d, i=12;
   char *buffer;
   std::cout << “Hello world!” << std::endl;
   // prints “Hello world!” to stdout

   std::cout << 10 << std::endl;
   // prints 10 to stdout

   std::cin >> d; // reads from stdin and
                  // assigns the value to d
   using namespace std;
   cout << “i = " << i << endl;
   // multiple writes through chain operations

   cin >> d >> buffer;
   // multiple reads through chain operations

   cin >> std::ws;  // eats white space
   return 0;
}
```

Input / Output στη C++ - iostream (3/3)

```
#include <iostream>
class Point {
     friend std::ostream& operator << (std::ostream& os, const Point& p);
     friend std::istream& operator >> (std::istream& is, Point& p);
     //friends so as to have access to the private members x and y
private:
     int x, y;
public:
     Point(int x = 0, int y = 0) : x(x), y(y) {}
};

//Output should never change the object, so const reference
std::ostream& operator << (std::ostream& os, const Point& p)
     { return os << "(" << p.x << "," << p.y << ")"; }

//Input should always change the object, so non-const reference
std::istream& operator >> (std::istream& is, Point& p)
     { return is >> p.x >> p.y; }

int main() {
     Point p;
     std::cin >> p;
     std::cout << "Point p is " << p << std::endl;
     return 0;
}
```

Input / Output στη C++ - fstreams (1/2)

- Ο χειρισμός αρχείων με τα *iostreams* είναι πολύ πιο εύκολος και πιο ασφαλής απ' ότι στη C
- Χρησιμοποιούμε αντικείμενα των τύπων ***ofstream*** (**o**utput **f**ile **stream**), ***ifstream*** (**i**nput **f**ile **stream**) και ***fstream*** (γενικό file stream)
- Οι τύποι αυτοί δηλώνονται στο header *fstream*
- Τα αρχεία ανοίγουν αυτόματα στους constructors και κλείνουν στους destructors
  - Υπάρχουν φυσικά και συναρτήσεις για να ανοίξουμε και να κλείσουμε το αρχείο όποτε εμείς θέλουμε
- Και πάλι τα input και output operations γίνεται με τους operators << και >>

Input / Output στη C++ - fstreams (2/2)

- Τα ifstreams ανοίγουν ένα αρχείο για είσοδο ενώ τα ofstreams για έξοδο
- Τα fstreams μπορούν να ανοίξουν ένα αρχείο και για είσοδο και για έξοδο, ανάλογα με το mode που τους δίνουμε
- Γενικά μπορούμε να ανοίξουμε ένα αρχείο σε διάφορα modes
  - FILE *fp = fopen(“lala.txt”, “r”)
  - fstream in (“lala.txt”, std::ios::in);
    - Το ***ios::in*** είναι το mode για ανάγνωση
    - Μπορούμε να δώσουμε περισσότερα από ένα modes ως mode1 | mode2

```
#include <iostream>
#include <fstream>
#define SIZE 100

using namespace std;

int main(){
  char buf[SIZE];
  ifstream in(“in.txt”);
  ofstream out(“out.txt”);

  in >> buf; // reads until
             // whitespace
  out << buf << std::endl;

  in.getline(buf, SIZE);
  //reads SIZE characters
  out << buf << std::endl;

  return 0;
}
```

Input / Output στη C++ - Open modes

Flag	Function
ios::in	**Opens an input file.**
ios::out	**Opens an output file. When used for an ofstream without ios::app, ios::ate or ios::in, ios::trunc is implied.**
ios::app	**Opens an output file for appending.**
ios::ate	**Opens an existing file (either input or output) and seeks the end.**
ios::nocreate	**Opens a file only if it already exists. (Otherwise it fails.)**
ios::noreplace	**Opens a file only if it does not exist. (Otherwise it fails.)**
ios::trunc	**Opens a file and deletes the old file, if it already exists.**
ios::binary	**Opens a file in binary mode. Default is text mode.**

C++ Exceptions

- Τα exceptions αποτελούν ένα τρόπο αντίδρασης του προγράμματος μας σε εξαιρετικές καταστάσεις μεταφέροντας τη ροή σε ειδικά σημεία κώδικα, τους ***handlers***
- Όταν θέλουμε να ελέγξουμε ένα κομμάτι κώδικα για exceptions το βάζουμε μέσα σε ένα ***try block***
- Αν συμβεί exception η ροή μεταφέρεται στο handler, ο οποίος δηλώνεται μέσα ***catch blocks*** και πρέπει να είναι αμέσως μετά από το try block.
- Για να προκαλέσουμε ένα exception χρησιμοποιούμε το keyword ***throw*** μέσα από το try block (ή από κάποια συνάρτηση που καλείται μέσα στο try block).

C++ Exceptions

- Παραδείγματα

```
#include <iostream>
using namespace std;
int main() {
  try { throw 20; }
  catch(int e) {
    cout << "Exception:" << e << endl;
  }  // prints Exception:20
  return 0;
}
```

```
try {
   try { throw 5; }
   catch(int e) { throw 'c' } // throw new
   catch(...) { throw; }      //rethrow
}
catch(int e) { cout << "Int Exception"; }
catch(...) { cout << “Default Exception";}
// prints Default Exception
```

```
#include <iostream>
using namespace std;
int main() {
  try {
     std::string option;
     std::cin >> option;
     if (option == "int") throw 0;
     else if (option == "char") throw 'a';
     else throw "string";
  }
  catch(int e) {cout << "Int Exception:" << endl; }
  catch(char e) {cout << "Char Exception:" << endl; }
  catch(...) {cout << “Default Exception:" << endl; }
  return 0;
}
```

Exception Specifications (1/2)

- Στη δήλωση μιας συνάρτησης μπορούμε να περιορίσουμε τα exceptions που μπορεί άμεσα ή έμμεσα να προκαλέσει προσθέτοντας την κατάληξη *throws*
- Π.χ. int function(float param) ***throws(int, bool)***;
  - Τα μόνα exception που μπορεί να προκαλέσει αυτή η συνάρτηση είναι τύπου int και bool
  - Αν προκαλέσει (άμεσα ή έμμεσα) άλλο exception αυτό δε μπορεί να πιαστεί από κάποιο handler
- Αν το throw είναι κενό (δηλαδή throw()) τότε δεν επιτρέπεται κανένα exception, ενώ αν δεν υπάρχει καθόλου επιτρέπονται όλα τα exceptions
  - int function (int param) throw(); // no exceptions allowed
  - int function (int param); // all exceptions allowed

Exception Specifications (2/2)

- Όταν έχουμε κληρονομικότητα και υλοποιούμε μια virtual συνάρτηση του base class που προκαλεί συγκεκριμένα exceptions, τότε πρέπει το derived class να προκαλεί το πολύ τα ίδια (ή λιγότερα exceptions).
- Προσοχή: στο Visual Studio δεν υλοποιούνται πλήρως τα exception specifications.

```
struct X {
   virtual void f(void) throw(int,bool) {}
};
struct Y : public X {
   void f(void) throw(int, bool) {} // ok
};
struct Z : public X {
  void f(void) throw(int, bool, double) {}
  // compile error: looser throw specifier
};
```

```
#include <iostream>
using namespace std;
void function(void) throw(int) { throw true; }
int main() {
   try { function(); }
   catch(int) { cout << "int" << endl;}
   catch(bool) { cout << "bool" << endl;}
   catch(...) { cout << "default" << endl; }
   return 0;
}
// VS  : runs ok printing: bool
// gcc : ternimates printing: terminate called
//       after throwing an instance of 'bool'
```

Standard Exceptions (1/2)

- Η standard βιβλιοθήκη παρέχει μια base class για αντικείμενα που γίνονται throw σαν exceptions
- Ονομάζεται exception, δηλώνεται στο header ***<exception>*** και ανήκει στο *namespace std*
- Η κλάση αυτή έχει μια virtual συνάρτηση *const char* what() const* που μπορεί να γίνει overwritten στα derived classes ώστε να παρέχει μια περιγραφή του exception.

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

Standard Exceptions (2/2)

C++ Standard Library Exceptions	
Exception	**Περιγραφή**
bad_alloc	thrown by new on allocation failure
bad_cast	thrown by dynamic_cast when fails with a referenced type
bad_exception	thrown when an exception type doesn't match any catch
bad_typeid	thrown by typeid
ios_base::failure	thrown by functions in the iostream library

```
#include <iostream>
#include <exception>
int main() {
    try { int * myarray = new int[500000000]; } // most likely to not have that memory
    catch (std::bad_alloc&) { std::cout << "Error allocating memory." << std::endl; }
    return 0;
}
```