Η ΓΛΩΣΣΑ C++

Μάθημα 5:

Κλάσεις: Πίνακες, Δομές και Αντικείμενα

Δημήτρης Ψούνης

Περιεχόμενα Μαθήματος

Α. Θεωρία

1. Περισσότερα για τις Κλάσεις
   1. Εναλλακτικός τρόπος κατασκευής constructor
   2. Συναρτήσεις – Μέθοδοι inline
   3. Default ορίσματα συναρτήσεων
   4. Ο δείκτης this
2. Κλάσεις και Πίνακες
   1. Κλάσεις που περιέχουν πίνακες
   2. Πίνακες που περιέχουν αντικείμενα
   3. Δυναμικός Πίνακας Αντικειμένων
   4. Πίνακας δεικτών σε αντικείμενα
3. Κλάσεις και Δομές
   1. Σχέση Δομών (structs) με Κλάσεις
4. Κλάσεις και const
   1. Const Αντικείμενα
   2. Const – Δείκτες – Αντικείμενα
   3. Const – Αναφορές – Αντικείμενα
5. Κλάσεις και αντικείμενα
   1. Κλάσεις που περιέχουν αντικείμενα
   2. Παράδειγμα
   3. Σειρά εκτέλεσης constructors-destructors
   4. Γενικά περί μοντελοποίησης

Ασκήσεις

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

1. Εναλλακτικός τρόπος κατασκευής constructor

- Βλέπουμε έναν εναλλακτικό τρόπο για να ορίσουμε έναν κατασκευαστή με ορίσματα:
  - Χωρίζοντας στο βήμα αρχικοποίησης και το σώμα της συνάρτησης
- Π.χ. αν έχουμε μία κλάση dummy με δύο ιδιωτικά μέλη x και y, αντί να χρησιμοποιήσουμε τον τρόπο που μάθαμε:

```
dummy::dummy(int in_x, int in_y)
{
    x = in_x;
    y = in_y;
    cout<<“Constructing…”;
}
```

- Μπορούμε να χρησιμοποιήσουμε το εξής εναλλακτικό συντακτικό:

```
dummy::dummy(int in_x, int in_y):
x(in_x), y(in_y)
{
    cout<<“Constructing…”;
}
```

  - Γίνονται οι αρχικοποιήσεις των μελών και έπειτα εκτελούνται άλλες εντολές που έχουμε στον κατασκευαστή.

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

1. Εναλλακτικός τρόπος κατασκευής constructor

- Και ένα ολοκληρωμένο παράδειγμα:

```
/*cpp5.construction_with_initialization_syntax.cpp:
Εναλλακτικός τρόπος αρχικοποίησης μελών κλάσης
από το constructor */

#include <iostream>
using namespace std;

class dummy {
  public:
    dummy(int in_x, int in_y);
    void print();
  private:
    int x,y;
};
```

```
int main()
{
   dummy ob (3,4);

   ob.print();

   return 0;
}

dummy::dummy(int in_x, int in_y):
x(in_x), y(in_y)
{
   cout<<"Constructing..."<<endl;
}

void dummy::print()
{
   cout<<x<<" "<<y;
}
```

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

2. Συναρτήσεις - Μέθοδοι inline

- Μία προσθήκη της C++ σε σχέση με τη C είναι οι συναρτήσεις inline:

- **Μία συνάρτηση inline:**
  - Δεν δημιουργεί δικό της χώρο, δεν είναι πραγματική συνάρτηση
  - Ενσωματώνει τον κώδικά της στην συνάρτηση που την καλεί
  - Ορίζεται βάζοντας τη λέξη-κλειδί inline μπροστά από το πρωτότυπο της

- Π.χ. στο ακόλουθο πρόγραμμα:

```
/*cpp5.inline_function.cpp: Συνάρτηση
inline */

#include <iostream>
using namespace std;

inline int sqr(int x);
```

```
int main()
{
   cout<<sqr(4);
    return 0;
}
int sqr(int x)
{
   return x*x;
}
```

- Η συνάρτηση sqr δεν καλείται, δεν δημιουργείται ξεχωριστός χώρος για αυτήν.
- Αλλά ο κώδικάς της ενσωματώνεται στη main κατά το χρόνο μεταγλώττισης
  - Γλιτώνουμε έτσι τον χρόνο που απαιτείται για την κλήση μιας συνάρτησης (δημιουργία χώρου, μεταφορά ελέγχου, επιστροφή)

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

2. Συναρτήσεις – Μέθοδοι inline

- Ο ορισμός μιας συνάρτησης ως **inline είναι αίτημα προς τον μεταγλωττιστή**:
  - Ο μεταγλωττιστής μπορεί να το απορρίψει, π.χ. όταν η συνάρτηση περιέχει κάποιο βρόχο, switch, στατικές μεταβλητές ή είναι αναδρομική
- Αν η συνάρτηση είναι περίπλοκη, τότε ενδέχεται να κάνει το πρόγραμμα χειρότερο:
  - Π.χ. αν δηλώνονται μεταβλητές, τότε οι μεταβλητές αυτές θα δηλωθούν στην καλούσα συνάρτηση τόσες φορές όσες είναι και η κλήση της.
  - Με αποτέλεσμα το πρόγραμμα να γίνει πιο βαρύ και μεγαλύτερο.

Συμπερασματικά:

- Ορίζουμε μία συνάρτηση να είναι inline, μόνο εφόσον είναι πάρα πολύ απλή.
  - π.χ. να κάνει μια απλή πράξη και να επιστρέφει.

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

2. Συναρτήσεις – Μέθοδοι inline

- **Μία μέθοδος που ορίζεται στη δήλωση της κλάσης είναι αυτόματα inline**
  - Χωρίς να χρειάζεται να βάλουμε μπροστά τη λέξη κλειδί inline.

- Έτσι για παράδειγμα στην ακόλουθη δήλωση κλάσης:

```
class dummy {
  public:
    void set_x(int in_x) { x = in_x; }
    int get_x() const { return x; }
  private:
    int x;
};
```

  - Μέσα στην δήλωση της κλάσης ορίσαμε το σώμα των μεθόδων:
    - με αποτέλεσμα αυτές οι μέθοδοι να είναι inline
    - (παρόλο που δεν το γράψαμε ρητα)

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

2. Συναρτήσεις – Μέθοδοι inline

- Για να τηρήσουμε την ανεξαρτησία της δήλωσης της κλάσης από το σώμα των μεθόδων:
  - η καλύτερη προσέγγιση στο συνταντικό θεωρείται ότι είναι:
    - Να κρατήσουμε τη δήλωση της μεθόδου καθαρή
    - Να δηλώσουμε ότι είναι inline, όταν γράφουμε το σώμα της μεθόδου:

```
class dummy {
  public:
    void set_x(int in_x);
    int get_x() const;
  private:
    int x;
};

inline void dummy::set_x(int in_x)
{
   x = in_x;
}
inline int get_x() const
{
   return x;
}
```

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

3. Default ορίσματα συναρτήσεων

- Έχουμε τη δυνατότητα στη C++ να βάζουμε προκαθορισμένα ορίσματα:

- **Ένα προκαθορισμένο όρισμα (default argument) σε μια συνάρτηση:**
  - Ορίζει μία τιμή για το όρισμα, αν στην κλήση της συνάρτησης δεν βάλουμε τιμή σε αυτό.

- Βλέπουμε ένα παράδειγμα:

```
/*cpp5.default_parameters.cpp:
Προκαθορισμένα ορίσματα συναρτήσεων
*/
#include <iostream>
using namespace std;

int sum(int x1, int x2=0, int x3=0, int x4=0);
```

```
int main()
{
   cout<<sum(1,2,3,4)<<endl;
   cout<<sum(1,2,3)<<endl;
   cout<<sum(1,2)<<endl;
   return 0;
}
int sum(int x1, int x2, int x3, int x4)
{
   return x1+x2+x3+x4;
}
```

- Η δήλωση των default ορισμάτων γίνεται στο πρωτότυπο της συνάρτησης
- Στο σώμα της συνάρτησης δεν γράφουμε τις default τιμές.
- Σημείωση: Όταν ορίζουμε ένα προκαθορισμένο όρισμα, πρέπει όλα τα επόμενα ορίσματα να έχουν και αυτά default τιμή. Π.χ. δεν μπορούμε να γράψουμε int sum(int x=0, int y, int z)

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

3. Default ορίσματα συναρτήσεων

- Είναι αρκετά συνηθισμένο, ένας constructor να έχει default τιμές στα ορίσματα του.
- Βλέπουμε μια παραλλαγή της κλάσης σημείο που χρησιμοποιεί default τιμές στα ορίσματά του:

```
class point {
   public:
      point(int x=0, int y=0);
      …
   private:
      int x,y;
};

point::point(int x, int y)
{
   x=0;
   y=0;
}
```

- και μπορούμε να δηλώσουμε αντικείμενα π.χ με τις δηλώσεις

```
point a(1,2); //Κατασκευάζει το (1,2)
point a(1);   //Κατασκευάζει το (1,0)
point a;      //Κατασκευάζει το (0,0)
```

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

4. Ο δείκτης this

- **Κάθε αντικείμενο έχει ως κρυφό μέλος, το δείκτη this**
  - Ο δείκτης this δείχνει τη διεύθυνση του αντικειμένου

- Βλέπουμε ένα παράδειγμα όπου απλά τυπώνουμε τη διεύθυνση μέσω του this:

```
/* CPP5.this.cpp : Ο δείκτης this
*/

#include <iostream>
using namespace std;

class point {
  public:
    point(int in_x, int in_y);
    point *ret_this();
    void print();
  private:
    int x;
    int y;
};
```

```
int main()
{
   point ob(3,4);

   cout<<"Address: "<<&ob<<endl;
   cout<<"Address: “
         <<ob.ret_this()<<endl;

   return 0;
}
```

```
point::point(int in_x, int in_y)
{
   x=in_x;
   y=in_y;
}

point *point::ret_this()
{
   return this;
}

void point::print()
{
   cout<<"("<<x<<","<<y<<")";
}
```

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

4. Ο δείκτης this

- Βλέπουμε ακόμη ένα παράδειγμα:
  - Παρατηρήστε τη σύγκρουση ονόματος. Το όνομα x του ορίσματος επικρατεί του ονόματος x του ιδιωτικού μέλους.
  - Με το δείκτη this ορίζουμε την πρόσβαση στην μεταβλητή x του αντικειμένου.

```
/* CPP5.this_constructor.cpp :
this και constructor */
#include <iostream>
using namespace std;

class point {
  public:
    point(int x, int y);
    void print();
  private:
    int x;
    int y;
};
```

```
int main()
{
   point ob(3,4);

   ob.print();

   return 0;
}
```

```
point::point(int x, int y)
{
   this->x=x;
   this->y=y;
}

void point::print()
{
   cout<<"("<<x<<","<<y<<")";
}
```

  - Ο τρόπος αυτός προτιμάται από αρκετούς προγραμματιστές.
  - Δεν θα τον ακολουθήσουμε σε αυτά τα μαθήματα.

Α. Θεωρία

1. Περισσότερα για τις κλάσεις

4. Ο δείκτης this

- και λίγα λόγια για τις συγκρούσεις ονομάτων
  - Εξετάζουμε το ακόλουθο πρόγραμμα (δεν μεταγλωττίζεται για την ώρα)

```
/* cpp5.name_collisions.cpp :
Συγκρούσεις ονομάτων */

#include <iostream>
using namespace std;

class dummy {
  public:
    dummy() { x = 2; }
    void print(int x);
  private:
    int x; // class member = 2
};
```

```
int x=1; // global = 1

int main()
{
   dummy ob;
   int x=3;

   ob.print(x);

   return 0;
}
```

```
void dummy::print(int x)
// argument = 3
{
   int x = 4; // local = 4
   cout<<x;
}
```

- Πειραματιστείτε με το πρόγραμμα αφαιρώντας μεταβλητές, ώστε να καταλήξετε στην ιεραρχία της C++ στις συγκρούσεις ονομάτων
- Έπικρατεί η πιο ειδική χρήση: (Τοπική Μεταβλητή) > (Μεταβλητή Μέλος) > (Καθολική Μεταβλητή)

Α. Θεωρία

2. Κλάσεις και Πίνακες

1. Κλάσεις που περιέχουν πίνακες

- Ένας πίνακας είναι μια στατική δομή δεδομένων
  - που σε διαδοχικές θέσεις μνήμης αποθηκεύει μεταβλητές του ίδιου τύπου δεδομένων.
- Οι κλάσεις μπορούν να περιέχουν ως μέλη πίνακες.
- Ας δούμε και ένα απλό παράδειγμα:

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

#define N 3

class grades
{
 public:
   grades();
   int set_i(int pos, int val);
   int get_i(int pos) const;
 private:
   int array[N];
};
```

```
int main()
{
   grades my_grades;

   /* Εισαγωγή Δεδομένων */
   my_grades.set_i(0,5);
   my_grades.set_i(1,8);
   my_grades.set_i(2,7);

  /* Εκτύπωση δεδομένων */
  for (int i=0; i<N; i++)
     cout<<my_grades.get_i(i)<<endl;

   return 0;
}
```

```
grades::grades()
{
    for (int i=0; i<N; i++)
        array[i]=0;
}
int grades::set_i(int pos, int val)
{
   if (pos>=0 && pos<N)
        array[pos]=val;
   else cout<<“out of bounds”;
}
int grades::get_i(int pos) const
{
   if (pos>=0 && pos<N)
      return array[pos];
   cout<<“out of bounds”;
   return 0;
}
```

Ολοκληρωμένο το πρόγραμμα είναι το: CPP5.class_with_array.cpp

Α. Θεωρία

2. Κλάσεις και Πίνακες

2. Πίνακες που περιέχουν αντικείμενα

- Συχνά θα χρειαστεί να έχουμε και πίνακες αντικειμένων.
- Η αρχικοποίηση των αντικειμένων μπορεί να γίνει με χρήση constructors θέτοντας τα αντικείμενα σε άγκιστρα χωρισμένα με κόμματα

```
/*
CPP5.array_of_objects.cpp :
Πίνακας που περιέχει
αντικείμενα */

#include <iostream>
using namespace std;

#define N 3

class dummy {
  public:
    dummy();
    dummy(int in_x);
    int get_x() const;
  private:
    int x;
};
```

```
int main()
{
   dummy array[N] = {dummy(),
            dummy(3), dummy(5)};

   for (int i=0; i<N; i++)
      cout<<array[i].get_x()<<" ";

   return 0;
}
```

```
dummy::dummy()
{
   x=0;
}

dummy::dummy(int in_x)
{
   x=in_x;
}

int dummy::get_x() const
{
   return x;
}
```

Α. Θεωρία

2. Κλάσεις και Πίνακες

2. Πίνακες που περιέχουν αντικείμενα

- Τα αντικείμενα μπορούν να αρχικοποιούνται και εκτός της δήλωσης.
- Ο default constructor θα τρέξει για κάθε αντικείμενο και έπειτα μπορούμε να τροποποιήσουμε τα αντικείμενα με τους accessors.

```
/*
CPP5.array_of_objects_noinit.cpp : Πίνακας που περιέχει αντικείμενα χωρίς αρχικοποίηση */

#include <iostream>
using namespace std;

#define N 3

class dummy {
  public:
    dummy();
    void set_x(int in_x);
    int get_x() const;
  private:
    int x;
};
```

```
int main()
{
   dummy array[N];

   for (int i=0; i<N; i++)
      array[i].set_x(i*i);

   for (int i=0; i<N; i++)
      cout<<array[i].get_x()<<" ";

   return 0;
}
```

```
dummy::dummy()
{
   x=0;
}

void dummy::set_x(int in_x)
{
   x=in_x;
}

int dummy::get_x() const
{
   return x;
}
```

Α. Θεωρία

2. Κλάσεις και Πίνακες

3. Δυναμικός Πίνακας Αντικειμένων

- Βλέπουμε και την υλοποίηση του ίδιου πίνακα με δυναμική δέσμευση μνήμης:

```
/*
CPP5.dynamic_array_of_obj
ects.cpp : Δυναμικός
Πίνακας Αντικειμένων  */

#include <iostream>
using namespace std;

class dummy {
  public:
    dummy();
    void set_x(int in_x);
    int get_x() const;
  private:
    int x;
};
```

```
int main()
{
   dummy *array;
   int n=3;

   array = new dummy [n];
   if (!array) cout<<"Mem error";

   for (int i=0; i<n; i++)
      array[i].set_x(i*i);

   for (int i=0; i<n; i++)
      cout<<array[i].get_x()<<" ";

   delete [] array;

   return 0;
}
```

```
dummy::dummy()
{
   x=0;
}

void dummy::set_x(int in_x)
{
   x=in_x;
}

int dummy::get_x() const
{
   return x;
}
```

Α. Θεωρία

2. Κλάσεις και Πίνακες

4. Πίνακας δεικτών σε αντικείμενα

- Ο πίνακας δεικτών σε αντικείμενα θα παίξει ιδιαίτερο ρόλο όταν θα μελετήσουμε την κληρονομικότητα.
- Βλέπουμε ένα πρώτο παράδειγμα:

```
/*
CPP5.array_of_pointers_to_
objects.cpp */

#include <iostream>
using namespace std;

#define N 3

class dummy {
  public:
    dummy();
    void set_x(int in_x);
    int get_x() const;
  private:
    int x;
};
```

```
int main()
{
   dummy *array[N];

   for (int i=0; i<N; i++)
      array[i] = new dummy;

   for (int i=0; i<N; i++)
      array[i]->set_x(i*i);

   for (int i=0; i<N; i++)
      cout<<array[i]->get_x()<<" ";

   for (int i=0; i<N; i++)
      delete array[i];

   return 0;
}
```

```
dummy::dummy()
{
   x=0;
}

void dummy::set_x(int in_x)
{
   x=in_x;
}

int dummy::get_x() const
{
   return x;
}
```

Α. Θεωρία

3. Κλάσεις και Δομές

1. Σχέση Δομών (structs) με Κλάσεις

- Ήδη γίνεται αντιληπτό ότι οι δομές (structs) της C μοιάζουν αρκετά με τις κλάσεις.
- Στην πραγματικότητά, είναι ισοδύναμες!
- Η μόνη διαφορά είναι ότι τα μέλη και οι μέθοδοι, αν δεν ορίζονται διαφορετικά:
  - Στις κλάσεις είναι ιδιωτικά.
  - Στις δομές είναι δημόσια.
- Έτσι οι ακόλουθες δύο δηλώσεις είναι «ισοδύναμες»

```
class C {
      int y;
  public:
      int f();
};
```

```
struct C {
      int f();
  private:
      int y;
};
```

  - με την έννοια ότι μπορούμε έπειτα να ορίσουμε αντικείμενα της δομής και της κλάσης που έχουν ακριβώς την ίδια λειτουργικότητα.

- Πρακτικά οι δομές δεν χρησιμοποιούνται στην C++, μιας και αντικαθίσταται από τις κλάσεις.
  - Αλλά υπάρχουν στην C++ και μπορούν να χρησιμοποιηθούν όπως στην C.
- Παραμένουν χρήσιμες, π.χ. όταν θέλουμε να έχουμε μόνο δημόσια μέλη
- βλ. και «Γλώσσα C – Μάθημα 13: Δομές»

Α. Θεωρία

4. Κλάσεις και const

1. Const Αντικείμενα

- Ένα αντικείμενο μπορεί να δηλωθεί ως const
  - Το αντικείμενο πρέπει να έχει constructor
  - Το αντικείμενο θα πρέπει να αρχικοποιηθεί με τον constructor και έπειτα δεν μπορεί να τροποποιηθεί ξανα (σφάλμα μεταγλώττισης)

```
/* cpp5.const_object.cpp
Const Αντικείμενα */

#include <iostream>
using namespace std;

class dummy {
public:
   dummy (int in_x);
   int get_x();
   void set_x(int in_x);
private:
   int x;
};
```

```
int main()
{
   const dummy ob(3);

   //ob.set_x(4); //error

   return 0;
}
```

```
dummy::dummy(int in_x)
{
   x=in_x;
}

int dummy::get_x()
{
   return x;
}

void dummy::set_x(int in_x)
{
   x=in_x;
}
```

Α. Θεωρία

4. Κλάσεις και const

2. Const – Δείκτες - Αντικείμενα

- Η σχέση const με τους δείκτες είναι ίδια όπως στη C:
- Περίπτωση 1: Δείκτης προς σταθερό αντικείμενο

```
const class_name *ptr
```

  - Το αντικείμενο δεν μπορεί να τροποποιηθεί
    - Χρήσιμο για να περάσουμε ένα όρισμα δείκτη σε αντικείμενο σε μια συνάρτηση
    - και δεν θέλουμε να τροποποιηθεί το αντικείμενο στη συνάρτηση.
  - Ο δείκτης μπορεί να αλλάξει (προς άλλο const αντικείμενο)
- Περίπτωση 2: Σταθερός δείκτης προς αντικείμενο

```
class_name * const ptr
```

  - Το αντικείμενο μπορεί να τροποποιηθεί
  - Ο δείκτης δεν μπορεί να δείξει αλλού
- Περίπτωση 3: Σταθερός δείκτης προς σταθερό αντικείμενο

```
const class_name * const ptr
```

  - Το αντικείμενο δεν μπορεί να τροποποιηθεί
  - Ο δείκτης δεν μπορεί να τροποποιηθεί

Α. Θεωρία

4. Κλάσεις και const

2. Const – Δείκτες - Αντικείμενα

- Βλέπουμε ένα παράδειγμα για την ενδιαφέρουσα περίπτωση (1):

```
/*
cpp5.pointer_to_const_obj
ect.cpp */

#include <iostream>
using namespace std;

class dummy {
public:
   dummy (int in_x);
   int get_x() const;
   void set_x(int in_x);
private:
   int x;
};
```

```
void f(const dummy *p);

int main()
{
   dummy ob(3);

   f(&ob);

   return 0;
}

dummy::dummy(int in_x)
{
   x=in_x;
}
```

```
int dummy::get_x() const
{
   return x;
}

void dummy::set_x(int in_x)
{
   x=in_x;
}

void f(const dummy *p)
{
   cout<<p->get_x();
   // p->set_x(2); // error
}
```

- Σημαντικό ότι σε ένα const αντικείμενο στο οποίο έχουμε πρόσβαση μέσω δείκτη, μπορούμε να καλέσουμε μόνο συναρτήσεις που είναι const.

Α. Θεωρία

4. Κλάσεις και const

3. Const – Αναφορές - Αντικείμενα

- Παρόλο που υπάρχουν δυνατότητες:
  - Δημιουργίας const αναφορών
- Μένουμε στην ενδιαφέρουσα περίπτωση, όπου περνάμε ως όρισμα σε συνάρτηση μία αναφορά σε ένα αντικείμενο που δεν αλλάζει (όπως στον copy constructor):

```
/*
cpp5.reference_to_const_object.cpp */

#include <iostream>
using namespace std;

class dummy {
public:
   dummy (int in_x);
   int get_x() const;
   void set_x(int in_x);
private:
   int x;
};
```

```
void f(const dummy &ref);

int main()
{
   dummy ob(3);

   f(ob);

   return 0;
}

dummy::dummy(int in_x)
{
   x=in_x;
}
```

```
int dummy::get_x() const
{
   return x;
}

void dummy::set_x(int in_x)
{
   x=in_x;
}

void f(const dummy &ref)
{
   cout<<ref.get_x();
   // ref.set_x(2); // error
}
```

- Επίσης ισχύει ότι έχουμε πρόσβαση μόνο σε συναρτήσεις που δεν αλλάζουν το αντικείμενο.

Α. Θεωρία

5. Κλάσεις που περιέχουν αντικείμενα

1. Θέματα μοντελοποίησης

- Είναι πάρα πολύ συχνό να έχουμε μία κλάση που περιέχει αντικείμενα άλλων κλάσεων.
- Όταν μοντελοποιούμε προβλήματα του πραγματικού κόσμου, αυτό θα είναι ιδιαίτερα συχνό.
- Παράδειγμα:
  - Μοντελοποιούμε μία τριόρωφη πολυκατοικία.
    - Η πολυκατοικία **έχει** 3 ορόφους
    - Κάθε όροφος **έχει** 2 διαμερίσματα
    - Κάθε διαμέρισμα έχει ένα πλήθος από ανθρώπους που είναι μέσα σε αυτό.
  - Θα κατασκευάσουμε ένα «χαφιέ» που να μας λέει πόσα άτομα είναι στην πολυκατοικία.

- Δίνουμε έμφαση στη **σχέση «έχει»**
  - Στην μοντελοποίηση αυτό θα σχετίζεται με μία κλάση που έχει αντικείμενα.
- π.χ.:
  - Ένας σκύλος έχει 4 πόδια
  - Ένα αυτοκίνητο έχει τιμόνι, 4 ρόδες, μηχανή
  - κ.λπ.

Α. Θεωρία

5. Κλάσεις που περιέχουν αντικείμενα

2. Παράδειγμα

- Ξεκινάμε με το διαμέρισμα (flat):

```
class flat {
  public:
    flat();
    ~flat();
    void set_people(int in_people);
    int get_people() const;
  private:
    int people;
};
```

```
flat::flat()
{
    people=0;
    cout<<"Constructing flat..."<<endl;
}
flat::~flat()
{
    cout<<"Destructing flat..."<<endl;
}
void flat::set_people(int in_people)
{
    people = in_people;
}
int flat::get_people() const
{
    return people;
}
```

Α. Θεωρία

5. Κλάσεις που περιέχουν αντικείμενα

2. Παράδειγμα

- Συνεχίζουμε με τον όροφο που περιέχει 2 διαμερίσματα:

```
class storey {
  public:
    storey();
    ~storey();
    void set_people(int in_flat, int in_people);
    int get_people() const;
  private:
    flat flats[2];
};
```

```
storey::storey()
{
    cout<<"Constructing storey..."<<endl;
}
storey::~storey()
{
    cout<<"Destructing storey..."<<endl;
}
void storey::set_people(int in_flat, int in_people)
{
    flats[in_flat].set_people(in_people);
}
int storey::get_people() const
{
    return flats[0].get_people()+flats[1].get_people();
}
```

Α. Θεωρία

5. Κλάσεις που περιέχουν αντικείμενα

2. Παράδειγμα

- Και με την πολυκατοικία (building) που περιέχει 2 όροφους:

```
class building {
  public:
    building();
    ~building();
    void set_people(int in_floor, int in_flat, int
in_people);
    int get_people() const;
  private:
    storey storeys[2];
};
```

```
building::building()
{
    cout<<"Constructing building..."<<endl;
}
building::~building()
{
    cout<<"Destructing building..."<<endl;
}
void building::set_people(int in_storey, int in_flat,
int in_people)
{
    storeys[in_storey].set_people(in_flat, in_people);
}
int building::get_people() const
{
    return
storeys[0].get_people()+storeys[1].get_people();
}
```

Α. Θεωρία

5. Κλάσεις που περιέχουν αντικείμενα

3. Σειρά εκτέλεσης constructors - destructors

- Η σειρά εκτέλεσης των κατασκευαστών είναι «από μέσα προς τα έξω»
  - Πρώτα το αντικείμενο που περιέχεται,
  - μετά το αντικείμενο που το περιέχει κ.ο.κ.
- Η σειρά εκτέλεσης των καταστροφέων είναι αντίστροφη της σειράς των κατασκευαστών.
- Βλέπουμε και τη σειρά εκτέλεσης στο παράδειγμά μας:

```
int main()
{
   building b;
   b.set_people(0,0,3);
   b.set_people(0,1,2);
   b.set_people(1,0,2);
   b.set_people(1,1,1);
   cout<<endl;
   cout<<"People in building: "<<b.get_people();
   cout<<endl<<endl;
   return 0;
}
```

Το πρόγραμμα είναι CPP5.class_with_objects.cpp

```
Constructing flat...
Constructing flat...
Constructing storey...
Constructing flat...
Constructing flat...
Constructing storey...
Constructing building...

People in building: 8

Destructing building...
Destructing storey...
Destructing flat...
Destructing flat...
Destructing storey...
Destructing flat...
Destructing flat...
```

Α. Θεωρία

5. Κλάσεις που περιέχουν αντικείμενα

4. Γενικά περί μοντελοποίησης

- Με το τελευταίο παράδειγμα, βλέπουμε ότι με τα προγραμματιστικά εργαλεία που έχουμε μάθει ήδη, μπορούμε να μοντελοποιήσουμε περίπλοκες καταστάσεις του πραγματικού κόσμου.
  - Η μοντελοποίηση και η οργάνωση των δεδομένων μας, είναι πολύ σημαντικά θέματα σε μεγάλα προγράμματα.

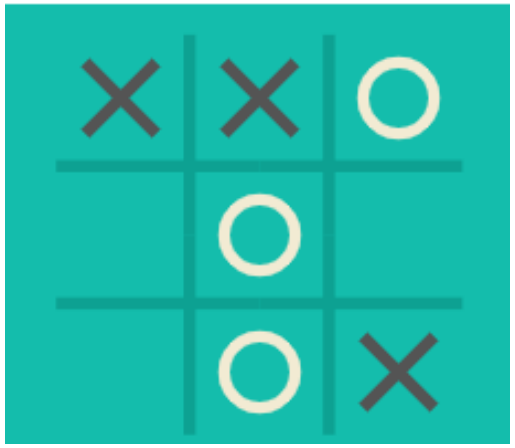
- Πρέπει όμως να διαχωρίσουμε στο μυαλό μας ότι:
  - **Μία γλώσσα προγραμματισμού είναι ένα εργαλείο σύνταξης** (κάτι σαν την γραμματική και το συντακτικό μιας πραγματικής γλώσσας)
    - Έτσι όταν μαθαίνουμε μια γλώσσα εστιάζουμε στη γλώσσα και στα τεχνικά χαρακτηριστικά της
  - Το τι πραγματικά θα κάνουμε με μια γλώσσα προγραμματισμού είναι αντικείμενο της **ανάλυσης συστημάτων**
    - Π.χ. πως θα οργανώσουμε τα δεδομένα μας, τη λειτουργικότητα, πως θα αλληλεπιδρούν μεταξύ τους τα στοιχεία κ.λπ.
    - και υπάρχουν ειδικές γλώσσες περιγραφής προδιαγραφών, όπως π.χ. η UML.

- Στα μαθήματα αυτά, θα βλέπουμε θέματα μοντελοποίησης στις ασκήσεις.
  - Αλλά κάποιος θα πρέπει να μελετήσει εκτός των γλωσσών προγραμματισμού, και τις τεχνικές ανάλυσης, ώστε να μπορεί να κατασκευάσει ολοκληρωμένα προγράμματα.

Β. Ασκήσεις

Άσκηση 1: Παιχνίδι: Tic Tac Toe (γνωστό και ως ΧΟΧ)

Θα υλοποιήσουμε μία κλάση που θα υλοποιεί το παιχνίδι tic-tac-toe (google it)



Συμβολική Σταθερά: N (να αρχικοποιείται σε 3)

Περιγραφή Κλάσης:

- Μέλος: Πίνακας 3x3 χαρακτήρων (να αρχικοποιούνται οι θέσεις σε '-')
- Μέθοδος play(x,y,c) παίρνει συντεταγμένες τετραγώνου και παίκτη (X ή O) και πραγματοποιεί την κίνηση (να γίνεται έλεγχος ότι μπορεί να γίνει η κίνηση)
- Μέθοδος check_win() επιστρέφει τον νικητή αν υπάρχει, και '-' αν δεν υπάρχει νικητής
- Μέθοδος print() εκτυπώνει το πλαίσιο

main:

- Επαναληπτικά:
  - Τυπώνει το πλαίσιο, τυπώνει ποιος παίζει και διαβάζει την επόμενη κίνηση.
  - Πραγματοποιεί την κίνηση και τυπώνει τον νικητή (αν υπάρχει)

Β. Ασκήσεις

Άσκηση 2: Παιχνίδι: Επέκταση του tic-tac-toe

Μεταφέρετε τον έλεγχο του ποιος παίκτης παίζει στην κλάση

Προσθέστε στην κλάση:

- Ορίστε ένα μέλος player (χαρακτήρας: X ή O)
- Μία μέθοδο next_player (αλλάζει το χαρακτήρα)
- Μία μέθοδο get_player (επιστρέφει τον παίκτη που παίζει)

Τροποποιήστε τη main:

- Ώστε να παίρνει τον παίκτη που παίζει από την κλάση

Β. Ασκήσεις

Άσκηση 3: Stranger Things..

Ένα hive, μάθαμε πρόσφατα, ότι:

- Περιέχει το πολύ έναν demigorgon (σιχαμερό τέρας)
  - Έχει κάποια χαρακτηριστικά (ύψος = 5μ) (βάρος = 500 κιλά)
  - Έχει δείκτη υγείας (health) που αρχικοποιείται σε 10000 πόντους.
  - Επιτίθεται (attack) κάνοντας ζημιά (τυχαία από 300-500 πόντοι)
- Περιέχει πολλά demidogs (βρωμερά σκυλιά που μοιάζουν με φυτά)
  - Είναι τυχαία από 10 έως 50.
  - Κάθε ένα έχει ένα δείκτη υγείας (health) που αρχικοποιείται σε 100 πόντους.
  - Κάθε demidog επιτίθεται (attack) κάνοντας ζημιά (τυχαία από 10-30 πόντοι)

Το hive:

- Έχει κατασκευαστή, ώστε να δεσμεύει δυναμικά:
  - Τον demigorgon (εφόσον υπάρχει)
  - Τα demidogs τα οποία αποθηκεύονται σε έναν δυναμικό πίνακα (το πλήθος τους είναι όρισμα)
- Επιτίθεται, ενεργοποιώντας τις επιθέσεις των τεράτων που υπάρχουν (τυπώνει τις επιμέρους επιθέσεις και επιστρέφει το συνολικό πλήθος πόντων ζημιάς που προξένησε)

Η main:

- Κατασκευάζει ένα hive που έχει demigorgon.
- Ενεργοποιεί την επίθεση τυπώνοντας το συνολικό πλήθος των πόντων ζημιάς που προκάλεσε.