ΜΑΘΗΜΑ 21

ΚΛΑΣΕΙΣ: ΠΙΝΑΚΕΣ ΔΟΜΕΣ ΚΑΙ ΑΝΤΙΚΕΙΜΕΝΑ

Γιώργος Διάκος - Full Stack Developer

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;
}
```

Ολοκληρωμένο το πρόγραμμα είναι το: CPP5.class_with_array.cpp

```
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;
}
```

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;
}
```

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;
}
```

2. Κλάσεις και Πίνακες

3. Δυναμικός Πίνακας Αντικειμένων

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

```
/*
CPP5.dynamic_array_of_objects.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;
}
```

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;
}
```

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

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;
}
```

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

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.

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_o
bject.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.

ΤΕΛΟΣ ΜΑΘΗΜΑΤΟΣ

ΚΛΑΣΕΙΣ: ΠΙΝΑΚΕΣ ΔΟΜΕΣ ΚΑΙ ΑΝΤΙΚΕΙΜΕΝΑ

Γιώργος Διάκος - Full Stack Developer