ΜΑΘΗΜΑ 20

ΚΛΑΣΕΙΣ: ΠΙΝΑΚΕΣ ΔΟΜΕΣ ΚΑΙ ΑΝΤΙΚΕΙΜΕΝΑ

Γιώργος Διάκος - Full Stack Developer

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.

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;
}
```

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)

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.

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)

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;
}
```

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)

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)
```

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<<")";
}
```

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.

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)

ΤΕΛΟΣ ΜΑΘΗΜΑΤΟΣ

ΚΛΑΣΕΙΣ: ΠΙΝΑΚΕΣ ΔΟΜΕΣ ΚΑΙ ΑΝΤΙΚΕΙΜΕΝΑ

Γιώργος Διάκος - Full Stack Developer