ΜΑΘΗΜΑ 18

ΚΛΑΣΕΙΣ ΚΑΙ ΑΝΑΦΟΡΕΣ 2/2

Γιώργος Διάκος - Full Stack Developer

3. Κατασκευαστής αντιγράφου

2. Ορισμός Κατασκευαστή Αντιγράφου (copy constructor)

- Για την αποφυγή παρόμοιων προβλημάτων,
- Μπορούμε να ορίσουμε έναν ειδικό κατασκευαστή, τον κατασκευαστή αντιγράφου.

- Ο **κατασκευαστής αντιγράφου καλείται αυτόματα** όταν:
  1. Διοχετεύουμε ως όρισμα μέσω τιμής (by value) ένα αντικείμενο
  2. Επιστρέφουμε μέσω τιμής (by value) ένα αντικείμενο
  3. Γίνεται δήλωση και αρχικοποίηση ενός αντικειμένου, μέσω ενός άλλου αντικειμένου

- Η δήλωση του κατασκευαστή αντιγράφου γίνεται ως εξής:

```
class_name(const class_name &ob)
```

  - Τον καλεί το νέο αντικείμενο (αντίγραφό) που κατασκευάζεται
  - Το όρισμα που δέχεται είναι το αντικείμενο μέσω του οποίου αρχικοποιείται.

- Θα δούμε αναλυτικά τις τρεις περιπτώσεις

3. Κατασκευαστής αντιγράφου

3. Κλήση όταν διοχετεύεται αντικείμενο σε συνάρτηση μέσω τιμής

Περίπτωση 1: Διοχέτευση αντικειμένου σε συνάρτηση μέσω τιμής

- Στο ακόλουθο παράδειγμα
  - Κατασκευάζουμε τον copy constructor στην κλάση
  - Βλέπουμε τι γίνεται όταν διοχετεύουμε αντικείμενο της κλάσης μέσω τιμής.

```
/*cpp4.copy_constructor_arg_by_value.cpp: Copy Constructor: Όρισμα μέσω τιμής */
#include <iostream>
using namespace std;

class dummy {
  public:
    dummy();
    dummy(const dummy &ob);
    ~dummy();
    int x;
};

void f(dummy arg);
```

3. Κατασκευαστής αντιγράφου

3. Κλήση όταν διοχετεύεται αντικείμενο σε συνάρτηση μέσω τιμής

```
int main()
{
   dummy d;

   cout<<"Main: Before calling f"<<endl;
   f(d);
   cout<<"Main: After calling f"<<endl;

   return 0;
}

dummy::dummy()
{
   cout<<"Constructing..."<<endl;
   x=0;
}
```

```
dummy::dummy(const dummy &ob)
{
   cout<<"Copy constructor..."<<endl;
   x=ob.x;
}

dummy::~dummy()
{
   cout<<"Destructing..."<<endl;
}

void f(dummy arg)
{
   cout<<"In function..."<<endl;
}
```

3. Κατασκευαστής αντιγράφου

3. Κλήση όταν διοχετεύεται αντικείμενο σε συνάρτηση μέσω τιμής

- Το πρόγραμμα εκτυπώνει:

```
Constructing...
Main: Before calling f
Copy constructor...
In function...
Destructing (x=10)
Main: After calling f
Destructing (x=0)
```

  - Στην main:
    - Κατασκευάζεται το αντικείμενο x
    - Καλείται η συνάρτηση f
      - Κατασκευάζεται το αντίγραφο μέσω του copy constructor
        - καλείται από το αντίγραφο (ob) με όρισμα το αντικείμενο (x)
        - Σαν να τρέχει η εντολή ob(x)
      - Διοχετεύεται ως όρισμα
      - Ολοκληρώνεται η εκτέλεση της συνάρτησης
      - Καταστρέφεται το αντίγραφο
    - Ολοκληρώνεται η main
    - Καταστρέφεται το αντικείμενο x

3. Κατασκευαστής αντιγράφου

4. Επιστροφη αντικειμένου από συνάρτηση μέσω τιμής

Περίπτωση 2: Επιστροφή αντικειμένου από συνάρτηση μέσω τιμής

- Στο ακόλουθο παράδειγμα
  - Κατασκευάζουμε τον copy constructor στην κλάση
  - Βλέπουμε τι γίνεται όταν επιστρέφουμε αντικείμενο της κλάσης μέσω τιμής.

```
/*cpp4.copy_constructor_return_ob.cpp: Copy Constructor: Επιστροφή αντικειμένου */
#include <iostream>
using namespace std;

class dummy {
  public:
    dummy();
    dummy(const dummy &ob);
    ~dummy();
    int x;
};

dummy f();
```

3. Κατασκευαστής αντιγράφου

4. Επιστροφη αντικειμένου από συνάρτηση μέσω τιμής

```
int main()
{
   cout<<"Main: Before calling f"<<endl;
   f();
   cout<<"Main: After calling f"<<endl;

   return 0;
}

dummy::dummy()
{
   cout<<"Constructing..."<<endl;
   x=0;
}
```

```
dummy::dummy(const dummy &ob)
{
   cout<<"Copy constructor..."<<endl;
   x=ob.x;
}

dummy::~dummy()
{
   cout<<"Destructing..."<<endl;
}

dummy f()
{
   dummy ob;
   cout<<"In function..."<<endl;
   return ob;
}
```

3. Κατασκευαστής αντιγράφου

4. Επιστροφη αντικειμένου από συνάρτηση μέσω τιμής

- Το πρόγραμμα εκτυπώνει:

```
Main: Before calling f
Constructing...
In function...
Destructing (x=0)...
Main: After calling f
```

- Δεν βλέπουμε τον copy constructor!

Ο onlineGDB έχει υλοποιήσει μία βελτιστοποίηση ειδικά για την περίπτωση αυτή (αναφέρεται ως: return value optimization)

- και δεν κατασκευάζει αντίγραφο.
- Ωστόσο σε άλλους μεταγλωττιστές ενδέχεται αυτή η βελτιστοποίηση να μην έχει υλοποιηθεί.

- Ωστόσο σε άλλο μεταγλωττιστή είναι δυνατόν να δούμε κλήση copy constructor στην περίπτωση αυτή.
  - Μετά την κλήση της συνάρτησης, θα βλέπαμε την κατασκευή ενός προσωρινού αντικειμένου το οποίο θα ήταν το αποτέλεσμα της κλήσης της συνάρτησης.

3. Κατασκευαστής αντιγράφου

5. Δήλωση και αρχικοποίηση μέσω άλλου αντικειμένου

Περίπτωση 3: Δήλωση και αρχικοποίηση μέσω άλλου αντικειμένου

- Στο ακόλουθο παράδειγμα
  - Κατασκευάζουμε τον copy constructor στην κλάση
  - Βλέπουμε τι γίνεται όταν δηλώνουμε ένα αντικείμενο και το αρχικοποιούμε μέσω άλλου αντικειμένου.

```
/*cpp4.copy_constructor_object_declaration.cpp: Δήλωση αντικειμένου μέσω άλλου αντικειμένου */
#include <iostream>
using namespace std;

class dummy {
  public:
    dummy();
    dummy(const dummy &ob);
    ~dummy();
    int x;
};
```

3. Κατασκευαστής αντιγράφου

5. Δήλωση και αρχικοποίηση μέσω άλλου αντικειμένου

```
int main()
{
   dummy ob1;
   dummy ob2 = ob1; // dummy ob2(ob1);

   return 0;
}
```

```
dummy::dummy()
{
   cout<<"Constructing..."<<endl;
   x=0;
}

dummy::dummy(const dummy &ob)
{
   cout<<"Copy constructor..."<<endl;
   x=ob.x;
}

dummy::~dummy()
{
   cout<<"Destructing..."<<endl;
}
```

3. Κατασκευαστής αντιγράφου

5. Δήλωση και αρχικοποίηση μέσω άλλου αντικειμένου

- Το πρόγραμμα εκτυπώνει:

```
Constructing...
Copy constructor...
Destructing (x=10)...
Destructing (x=0)...
```

  - Στην main:
    - Κατασκευάζεται το αντικείμενο ob1
    - Η δήλωση dummy ob2 = ob1 καλεί τον copy constructor
      - καλείται από το ob2 με όρισμα το ob1
      - είναι ισοδύναμο με τη δήλωση:

```
dummy ob2(ob1);
```

    - Καταστρέφονται τα δύο αντικείμενα.

ΠΡΟΣΟΧΗ: Τα παρακάτω είναι διαφορετικά πράγματα:

- dummy ob2 = ob1; (δήλωση και αρχικοποίηση, καλείται ο copy constructor)
- ob2 = ob1; (ανάθεση, θα αντιμετωπιστεί με υπερφόρτωση του =)

3. Κατασκευαστής αντιγράφου

6. Παρατηρήσεις

- Ο copy constructor είναι μία ευκολία.
- Χρειάζεται πάντα να τον ορίζούμε;
  - ΌΧΙ, αν η κλάση μας είναι πολύ απλή, δεν χρειάζεται copy constructor
    - σκεφτόμαστε ότι η bit by bit αντιγραφή δεν θα δημιουργήσει προβλήματα.
  - Αν όμως κάνουμε στην κλάση δυναμική διαχείριση μνήμης
    - π.χ. έχουμε έναν δείκτη που δεσμεύει δυναμικά μνήμη για έναν πίνακα
    - τότε ο copy constructor είναι ΑΠΑΡΑΙΤΗΤΟΣ.
- Γιατί είναι το όρισμα const;
  - Με το const ορίζουμε ότι δεν πρέπει να πειραχτεί το αρχικό αντικείμενο
  - Πράγματι, η δουλειά του copy constructor είναι μόνο να πάρει πληροφορίες από το αντικείμενο που αντιγράφεται και όχι να το πειράξει.

ΑΣΚΗΣΗ 1 : ΣΥΝΑΡΤΗΣΗ SWAP

Υλοποίησε την Συνάρτηση swap ώστε να δέχεται δυο ορίσματα και να ανταλλάσει τις τιμές τους :

- Οι τιμές μπορούν να είναι :

1. Int ,
2. double,
3. Αντικείμενα της κλάσης σημείο (βλ. Lesson_15b << Εισαγωγή στις κλάσεις>> , άσκηση 2)

ΑΣΚΗΣΗ 2 : ΕΠΕΚΤΑΣΗ ΤΗΣ ΚΛΑΣΗΣ ARRAY

Επέκτεινε την κλάση ARRAY της άσκησης 2
του lesson_17: <<Κλάσεις και Δείκτες>> με κατασκευαστή αντιγράφου.

ΑΣΚΗΣΗ 3 : ΚΛΑΣΗ STRING

Κατασκεύασε μια κλάση (STRING) που να περιτυλίσσει την έννοια της συμβολοσειράς ως εξής :

- Να έχει ως μέλη εναν δυναμικό πίνακα (δείκτης) χαρακτήρων, καθώς και την διάστασταση του πίνακα.
- Να έχει κατασκευαστές :
- Default
- Μόνο με την συμβολοσειρά (να γίνεται δυναμική δέσμευση μνήμης όση και το μήκος της συμβολοσειράς εισόδου)
- Αντιγράφου
- Ο καταστροφέας να διαγράφει την μνήμη που έχει δεσμευτεί δυναμικά.
- Getter : Για την συμβολοσειρά και το μήκος
- Setter : Μόνο για την συμβολοσειρά

Η συνάρτηση main :

- Να κατασκευάζει μια συμβολοσειρά str1 , που αρχικοποιείται ως << This is a String >>
- Να την αντιγράφει στην συμβολοσειρά str2 μέσω του copy constructor.
- Να αλλάζει την str1 σε << This is a new string>>
- Να τυπώνει τις 2 συμβολοσειρές

ΤΕΛΟΣ ΜΑΘΗΜΑΤΟΣ

ΚΛΑΣΕΙΣ ΚΑΙ ΑΝΑΦΟΡΕΣ 2/2

Γιώργος Διάκος - Full Stack Developer