Η ΓΛΩΣΣΑ C++

Μάθημα 4:

Κλάσεις και Αναφορές

Δημήτρης Ψούνης

www.psounis.gr

Περιεχόμενα Μαθήματος

Α. Θεωρία

1. Αναφορές

1. Τι είναι αναφορά

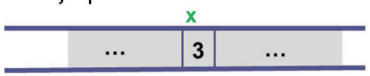
- Οι αναφορές είναι ένα νέο στοιχείο της C++

- Η **αναφορά** (reference) είναι ένα συνώνυμο (alias) μιας μεταβλητής.

- Η δήλωση μιας αναφοράς:
  - **Πρέπει** να συνοδευεται από την μεταβλητή, της οποίας θα είναι συνώνυμο.
  - Στην δήλωση, το & πρέπει να προηγείται του ονόματος της αναφοράς
  - Παράδειγμα μιας αναφοράς ref στην ακέραια μεταβλητή x:

```
int &ref = x;
```

  - Και πλέον όταν γράφουμε ref είναι σαν να γράφουμε x.
- Προσοχή!
  - Η αναφορά δεν είναι ανεξάρτητη από τη μεταβλητή στην οποία αναφέρεται.
  - Δεν έχει κάποια ανεξάρτητη διεύθυνση, κάτι που να μοιάζει με δείκτη, ή οτιδήποτε άλλο.
  - Σχηματικά το απεικονίζουμε:



- Κάποιοι προγραμματιστές προτιμούν να γράφουν την δήλωση ως: int& ref;

Α. Θεωρία

1. Αναφορές

2. Παράδειγμα χρήσης αναφοράς

- Βλέπουμε και ένα απλό παράδειγμα επίδειξης της χρήσης της αναφοράς:

```
/* cpp4.reference.cpp Χρήση αναφοράς */
#include <iostream>
using namespace std;

int main()
{
   int x = 3;
   int &r = x;

   cout<<"x="<<x<<" r="<<r<<endl;

   x=4;
   cout<<"x="<<x<<" r="<<r<<endl;

   r=5;
   cout<<"x="<<x<<" r="<<r<<endl;

   return 0;
}
```

Α. Θεωρία

1. Αναφορές

3. ...και το πιο συνηθισμένο λάθος!

- Ο ακόλουθος κώδικας δεν κάνει αυτό που θέλει ο προγραμματιστής (βλ. σχόλια):

```
/* cpp4.reference_mistake.cpp Συνηθισμένο λάθος στις αναφορές */
#include <iostream>
using namespace std;

int main()
{
   int x = 3, y=4;
   int &r = x;

   cout<<"r="<<r<<endl;

   r=y; // Προσπαθεί να αλλάξει την αναφορά σε άλλη μεταβλητή

   cout<<"x="<<x<<endl; // Άλλαξε όμως το x

   return 0;
}
```

- Προσοχή: Η αναφορά σχετίζεται αποκλειστικά με την μεταβλητή για την οποία δηλώθηκε αρχικά.

Α. Θεωρία

2. Αναφορές και Συναρτήσεις

1. Διοχέτευση ορίσματος μέσω τιμής (by value)

- Να κάνουμε μία κλάσική υπενθύμιση από τη C, για τον τρόπο που διοχετεύουμε ορίσματα σε συναρτήσεις:
- Ο 1ος τρόπος είναι μέσω τιμής, όπου ξέρουμε ότι οι αλλαγές που γίνονται στη συνάρτηση, δεν διατηρούνται έξω από αυτήν:

```
/*cpp4.byvalue.cpp: Πέρασμα ορίσματος μέσω
τιμής */

#include <iostream>
using namespace std;

void change(int vA);

int main()
{
   int a=1;
   cout<<"main: a="<<a<<endl;
   change(a);
   cout<<"main: a="<<a<<endl;

   return 0;
}

void change(int vA)
{
   vA=2;
   cout<<"change: vA="<<vA<<endl;
}
```

- Η έξοδος είναι:

```
main: a=1
change: vA=2
main: a=1
```

- Πολύ αναλυτική εξήγηση στο παράδειγμα, στο «Γλώσσα C: Μάθημα 8: Δείκτες»

Α. Θεωρία

2. Αναφορές και Συναρτήσεις

2. Διοχέτευση ορίσματος μέσω δείκτη (by pointer)

- Ο 2ος τρόπος είναι μέσω δείκτη, όπου ξέρουμε ότι οι αλλαγές που γίνονται στη συνάρτηση, διατηρούνται έξω από αυτήν:

```
/*cpp4.bypointer.cpp: Πέρασμα ορίσματος μέσω
δείκτη */

#include <iostream>
using namespace std;

void change(int *pA);

int main()
{
   int a=1;
   cout<<"main: a="<<a<<endl;
   change(&a);
   cout<<"main: a="<<a<<endl;

   return 0;
}

void change(int *pA)
{
   *pA=2;
   cout<<"change: *pA="<<*pA<<endl;
}
```

- Η έξοδος είναι:

```
main: a=1
change: *pA=2
main: a=2
```

- Στη C ο τρόπος αναφέρεται και «by reference», αλλά στη C++ αναφέρεται «by pointer»

Α. Θεωρία

2. Αναφορές και Συναρτήσεις

3. Διοχέτευση ορίσματος μέσω αναφοράς (by reference)

- Ο 3ος τρόπος είναι μέσω αναφοράς (νέος τρόπος)
- Ο ορισμός της συνάρτησης έχει αναφορά:

```
void change(int &rA)
{
   rA=2;
   cout<<"change: rA="<<rA<<endl;
}
```

- Παρατηρήσεις:
  - Διοχετεύοντας αναφορά, δεν δημιουργείται αντίγραφο του ορίσματος, άρα με έμμεσο τρόπο διοχετεύεται η ίδια η μεταβλητή.
  - Συνεπώς οι αλλαγές που γίνονται σε αυτήν διατηρούνται και εκτός της συνάρτησης.
  - Μέσα στη συνάρτηση, η χρηση της μεταβλητής είναι πιο «εύκολη» από την διαχείριση που κάνουμε όταν διοχετεύουμε το όρισμα μέσω δείκτη.

Α. Θεωρία
2. Αναφορές και Συναρτήσεις
3. Διοχέτευση ορίσματος μέσω αναφοράς (by reference)

- Ολοκληρωμένο το πρόγραμμα είναι:

```
/*cpp4.byreference.cpp: Πέρασμα ορίσματος μέσω αναφοράς */

#include <iostream>
using namespace std;

void change(int &rA);

int main()
{
   int a=1;
   cout<<"main: a="<<a<<endl;
   change(a);
   cout<<"main: a="<<a<<endl;

   return 0;
}

void change(int &rA)
{
   rA=2;
   cout<<"change: rA="<<rA<<endl;
}
```

- **Οι αναφορές λοιπόν κάνουν πιο εύκολη τη ζωή μας σε σχέση με τη διαχείριση δεικτών**:
  - Ένας καλός κανόνας είναι
    - **«Χρησιμοποίησε αναφορές όποτε μπορείς και δείκτες όποτε πρέπει»** ©

© www.geeksforgeeks.org

Α. Θεωρία
3. Κατασκευαστής αντιγράφου
1. Αντίγραφα Αντικειμένων

- Όταν διοχετεύουμε σε συνάρτηση όρισμα το οποίο είναι αντικείμενο, μέσω τιμής:
  - Δημιουργείται αντίγραφο του αντικειμένου (όπως γίνεται και με τις απλές μεταβλητές)
- Βλέπουμε ένα παράδειγμα:

```
/*cpp4.copy_arg_by_value.cpp: Δημιουργία αντιγράφου όταν αντικείμενο διοχετεύεται ως όρισμα */
#include <iostream>
using namespace std;

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

void f(dummy ob);

int main()
{
   dummy d;
      // καλείται ο constructor

   f(d);

   return 0;
}


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

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

void f(dummy ob)
{
   // το όρισμα είναι αντίγραφο

   // καταστρέφεται το αντίγραφο
}
```


Α. Θεωρία
3. Κατασκευαστής αντιγράφου
1. Αντίγραφα Αντικειμένων

- Εκτελώντας το πρόγραμμα έχουμε την εκτύπωση:

```
Constructing...
Destructing...
Destructing...
```

- Εξήγηση:
  - Main:
    - Δήλωση του αντικειμένου d (καλείται ο constructor).
  - Κλήση f(d):
    - Δημιουργείται αντίγραφο bit by bit (δεν καλείται constructor), αφού το όρισμα είναι μέσω τιμής.
    - Με την ολοκλήρωση της συνάρτησης, καταστρέφεται το αντίγραφο
  - Συνέχεια της main
    - Καταστρέφεται το αντικείμενο d

Αντίστοιχες καταστάσεις μπορεί να συμβούν π.χ. όταν:
- Επιστρέφουμε ένα αντικείμενο σε μία συνάρτηση μέσω τιμής (όχι δείκτη ή αναφορά)

Για τις περιπτώσεις αυτές (και άλλες), η C++ ορίζει μια προγραμματιστική ευκολία:
- Τον κατασκευαστή αντιγράφων (copy constructor)

Α. Θεωρία
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

Α. Θεωρία

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);
```


Α. Θεωρία

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;
}
```


Α. Θεωρία

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

Α. Θεωρία

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();
```

Α. Θεωρία

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;
}
```


Α. Θεωρία

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.

Α. Θεωρία

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;
};
```


Α. Θεωρία

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;
}
```

Α. Θεωρία

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 =)

Α. Θεωρία

3. Κατασκευαστής αντιγράφου

6. Παρατηρήσεις

- Ο copy constructor είναι μία ευκολία.
- Χρειάζεται πάντα να τον ορίζούμε;
  - ΟΧΙ, 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.

Β. Ασκήσεις

Άσκηση 1: Συνάρτηση swap

Υλοποιήστε τη συνάρτηση swap ώστε να δέχεται δύο ορίσματα και να ανταλλάσσει τις τιμές τους.
- Οι τιμές μπορούν να είναι:
  - int,
  - double,
  - αντικείμενα της κλάσης σημείο (βλ. μάθημα 2 «Εισαγωγή στις κλάσεις», άσκηση 2)

Β. Ασκήσεις

Άσκηση 2: Επέκταση της κλάσης ARRAY

Επεκτείνετε την κλάση ARRAY της άσκησης 2 του «Μάθημα 3: Κλάσεις και Δείκτες» με κατασκευαστή αντιγράφου.

Β. Ασκήσεις

Άσκηση 3: Κλάση STRING

Κατασκευάστε μία κλάση (STRING) που να περιτυλίσσει την έννοια της συμβολοσειράς ως εξής:

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

Η συνάρτηση main

- Να κατασκευάζει μία συμβολοσειρά str1, που αρχικοποιείται ως «This is a string»
- Να την αντιγράφει στη συμβολοσειρά str2 μέσω του copy constructor.
- Να αλλάζει την str1 σε «This is a new string»
- Να τυπώνει τις δύο συμβολοσειρές