ΜΑΘΗΜΑ 18

ΚΛΑΣΕΙΣ ΚΑΙ ΑΝΑΦΟΡΕΣ 1/2

Γιώργος Διάκος - Full Stack Developer

1. Αναφορές

1. Τι είναι αναφορά

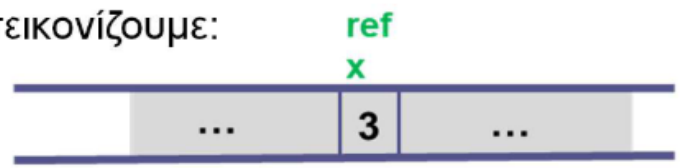
- Οι αναφορές είναι ένα νέο στοιχείο της C++

- 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;

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;
}
```

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.

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
```

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»

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.

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»** ©

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
}
```

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)

ΤΕΛΟΣ ΜΑΘΗΜΑΤΟΣ

ΚΛΑΣΕΙΣ ΚΑΙ ΑΝΑΦΟΡΕΣ 1/2

Γιώργος Διάκος - Full Stack Developer