



ΜΑΘΗΜΑ 16

ΚΛΑΣΕΙΣ ΚΑΙ ΔΕΙΚΤΕΣ

Γιώργος Διάκος - Full Stack Developer

1. Διαχείριση Μνήμης

1. Στατική και Δυναμική Δέσμευση Μνήμης

- Η διαχείριση μνήμης στη C++ γίνεται με παρόμοιο τρόπο με τη C.
- Τα παρακάτω είναι εντελώς αντίστοιχα:

- **Στατική Δέσμευση Μνήμης:**

- Είναι γνωστός εκ των προτέρων ο χώρος μνήμης που απαιτείται για τα στιγμιότυπα.
- Και έχει ήδη αποφασιστεί κατά το χρόνο μεταγλώττισης (π.χ. δηλώνουμε έναν πίνακα 10 θέσεων)
- Οι τοπικές μεταβλητές αποθηκεύονται στον χώρο της συνάρτησης (**στοίβα** – stack)

- **Δυναμική Δέσμευση Μνήμης:**

- Δεν είναι γνωστός εκ των προτέρων ο χώρος μνήμης που πρέπει να δεσμεύσει το πρόγραμμά μας.
- Αλλά αποφασίζεται κατά το χρόνο εκτέλεσης (π.χ. ο χρήστης αποφασίζει πόσες θέσεις θα έχει ένας πίνακας που χρησιμοποιεί το πρόγραμμα)
- Ο χώρος μνήμης που δεσμεύεται δυναμικά είναι κοινός για όλες τις συναρτήσεις (**σωρός** – heap)

ΣΤΑΤΙΚΗ ΔΕΣΜΕΥΣΗ ΜΝΗΜΗΣ



[This Photo](#) by Unknown author is licensed under [CC BY-NC](#).

1. Διαχείριση Μνήμης

2. Στατική Δέσμευση μνήμης για συνήθεις μεταβλητές

```
/* CPP3.local_variables.cpp */
```

```
#include <iostream>
using namespace std;
```

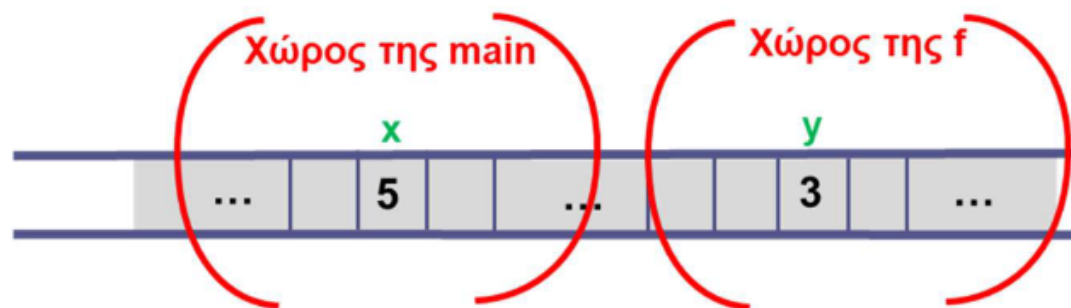
```
void f()
{
    int y=3;
    cout<<"inside f";
}
```

```
int main()
{
    int x=5;

    cout<<"inside main";
    f();

    return 0;
}
```

- Κάθε συνάρτηση έχει το δικό της χώρο μνήμης.
- Κάθε τοπική μεταβλητή αποθηκεύεται στο χώρο μνήμης της συνάρτησης στην οποία ανήκει.
- Σχηματικά, για το παράδειγμα:



- Αυτός ο χώρος μνήμης καλείται και στοίβα (stack), λόγω του τρόπου με τον οποίο έχει υλοποιηθεί η κλήση των συναρτήσεων σε μία στοίβα:



1. Διαχείριση Μνήμης

3. Στατική Δέσμευση μνήμης για Αντικείμενα

- Ένα αντικείμενο αποθηκεύεται στη μνήμη ως εξής:
 - Τα μέλη του βρίσκονται σε διαδοχικές θέσεις μνήμης
- Προσοχή, οι μέθοδοι βρίσκονται σε άλλο χώρο (είναι κοινές για όλα τα αντικείμενα της κλάσης)
- Ο τελεστής **sizeof** μπορεί να χρησιμοποιηθεί για να μετρήσουμε το πλήθος των bytes που χρησιμοποιεί ένα αντικείμενο.
- Παράδειγμα:

```
/* CPP3.sizeof.cpp
Μέγεθος αντικειμένου */
#include <iostream>
using namespace std;

class dummy {
public:
    int x;
    int y;
};
```

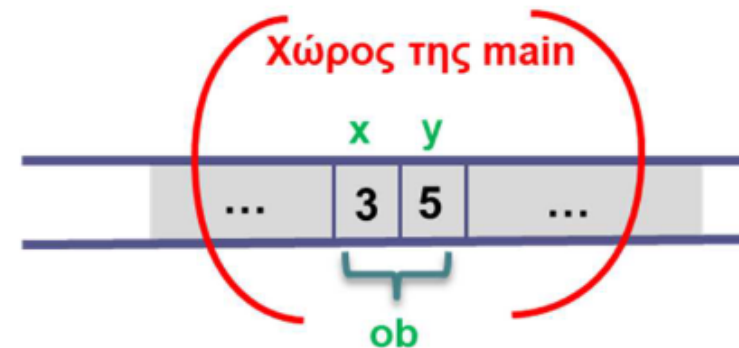
```
int main()
{
    dummy ob;

    ob.x = 3; ob.y = 5;

    cout<<sizeof ob<<endl;

    return 0;
}
```

- Θα τυπώσει 8 (δύο ακέραιοι 4bytes ο καθένας)
- Και η εικόνα της μνήμης:



ΔΥΝΑΜΙΚΗ ΔΕΣΜΕΥΣΗ ΜΝΗΜΗΣ



[This Photo](#) by Unknown author is licensed under [CC BY-NC](#).

2. Δυναμική Δέσμευση Μνήμης

1. Δείκτες (Υπενθύμιση από C)

- Ένας δείκτης είναι μια μεταβλητή που αποθηκεύει διευθύνσεις μεταβλητών

- Ένας δείκτης δηλώνεται ως εξής:

```
TΔ *pointer_name;
```

- Όπου TΔ είναι ένας οποιοσδήποτε τύπος δεδομένων (int, float, double κ.λπ.)
 - Π.χ. ένας δείκτης σε ακέραιο δηλώνεται με τη δήλωση:

```
int *ptr;
```

- Αποθηκεύουμε σε έναν δείκτη την διεύθυνση μιας μεταβλητής με τη δήλωση:

```
pointer_name = &variable;
```

- Ο τελεστής & επιστρέφει τη διεύθυνση της μεταβλητής στην οποία εφαρμόζεται.
 - Π.χ. αποθηκεύουμε στον δείκτη ptr τη διεύθυνση μιας μεταβλητής x με την εντολή:

```
ptr = &x;
```

- και λέμε «ο δείκτης ptr δείχνει στην μεταβλητή x»

2. Δυναμική Δέσμευση Μνήμης

1. Δείκτες (Υπενθύμιση από C)

```
/* CPP3.pointers.cpp */

#include <iostream>
using namespace std;

int main()
{
    int x=5;
    int *p = NULL;

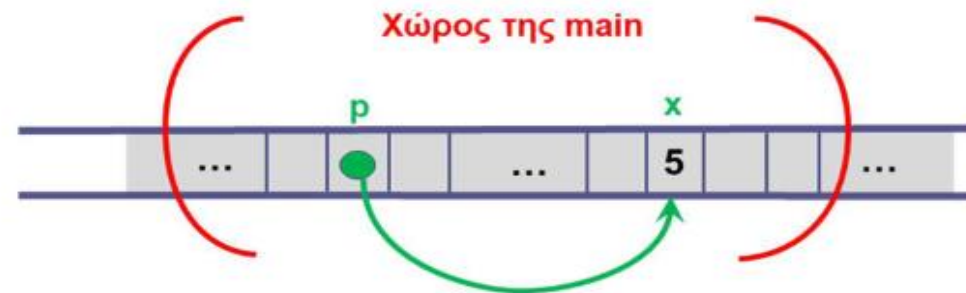
    p = &x;

    cout<<"x = "<<x<<" (address: "<<&x<<)"<<endl;
    cout<<"*p = "<<*p<<" (address: "<<p<<)"<<endl;
    return 0;
}
```

- Το πρόγραμμα δείχνει την βασική ιδιότητα:
 - Για ένα δείκτη (p) που «δείχνει» σε μία μεταβλητή ισχύουν:

```
p == &x //Διεύθυνση της x
*p == x // Τιμή της x
```

- Το NULL είναι μία ειδική τιμή
 - που συμβολίζει ότι ο δείκτης δεν δείχνει κάπου.
 - Είναι καλό ένας δείκτης που δεν δείχνει κάπου να έχει την τιμή NULL
 - Η τιμή του NULL είναι 0.
- Σχηματικά:



2. Δυναμική Δέσμευση Μνήμης

2. Οι τελεστές new και delete

- Ο τελεστής **new** χρησιμοποιείται για να δεσμεύσει δυναμικά χώρο μνήμης
 - Αντικαθιστά τη malloc της C.
 - Η σύνταξη του είναι:

```
ptr = new type;
```

- Δεσμεύει χώρο μνήμης για ένα αντικείμενου τύπου type **και επιστρέφει έναν δείκτη σε αυτόν το χώρο**
- Ο χώρος μνήμης που δεσμεύεται δυναμικά, είναι κοινός για όλες τις συναρτήσεις και ονομάζεται σωρός (heap)
- Προσοχή, αν αποτύχει η δέσμευση (δεν υπάρχει χώρος) επιστρέφει NULL
 - **και θα πρέπει πάντα να ελέγχουμε ότι η δέσμευση έγινε επιτυχημένα**
- Ο τελεστής **delete** χρησιμοποιείται για να απόδεσμευσει χώρο μνήμης που έχει δεσμευτεί δυναμικά
 - Αντικαθιστά τη free της C.
 - Η σύνταξη του είναι:

```
delete ptr;
```

2. Δυναμική Δέσμευση Μνήμης

2. Οι τελεστές new και delete (Έλεγχος Δέσμευσης Μνήμης)

- Πάντα πρέπει να κάνουμε έλεγχο αν η μνήμη που αιτηθήκαμε με τη new, έχει όντως δεσμευτεί.
- Βλέπουμε τρεις τρόπους για να «πιάσουμε» το πρόβλημα:
- Α' τρόπος: Ο πιο απλός, μιας και απλά ελέγχουμε την επιστρεφόμενη τιμή:

```
ptr = new type;  
if (ptr==NULL)  
{ .... do some action... }
```

- Β' τρόπος: Πιο ψαγμένος, ξέρουμε ότι το NULL είναι 0 (και για να είμαστε ακριβής (void *) 0)
 - Και ξέρουμε ότι το 0, είναι το λογικό ψευδές
 - και το !0 είναι το λογικό αληθές:

```
ptr = new type;  
if (!ptr)  
{ .... do some action... }
```

- Γ' τρόπος: Ο πιο σύνθετος και φαντεζί, αλλά ενσωματώνει τις δύο εντολές σε μία γραμμή:

```
if (!(ptr = new type))  
{ .... do some action... }
```

- Στις σημειώσεις θα χρησιμοποιήσουμε κυρίως το Β' τρόπο
- Και θα βγάζουμε απλά ένα ενημερωτικό μήνυμα (αργότερα θα μπορούμε να κάνουμε και περισσότερα πράγματα, με τις εξαιρέσεις της C++)

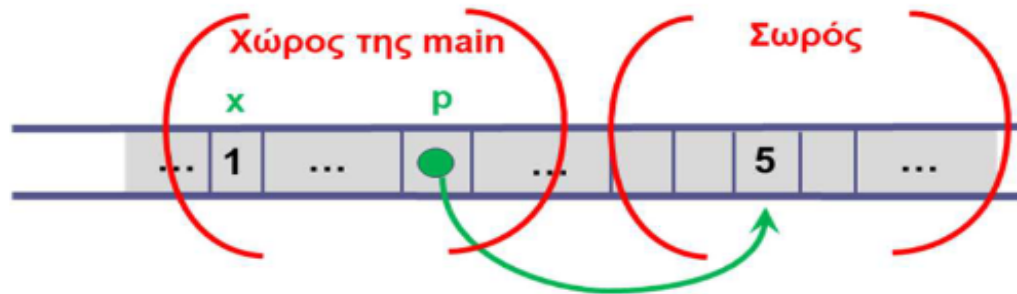
2. Δυναμική Δέσμευση Μνήμης

3. Δυναμική Δέσμευση για Συνήθεις Μεταβλητές

- Το παράδειγμα δείχνει τη **δέσμευση μνήμης για έναν ακέραιο αριθμό**, τόσο στατικά όσο και δυναμικά:

Υπενθυμίσεις από τη C:

- Η x δεσμεύει χώρο μνήμης μέσα στο χώρο μνήμης της συνάρτησης
- Ο χώρος που δεσμεύεται δυναμικά μέσω της p , είναι ένας ξεχωριστός χώρος, κοινός για όλο το πρόγραμμα
 - (λέγεται και **σωρός** – heap)
- Σχηματικά:



```
#include <iostream> // CPP3.dynamic_variable.cpp
using namespace std;

int main()
{
    int x; // Μεταβλητή που δεσμεύει στατικό χώρο
    int *p; // Δείκτης σε ακέραιο

    p = new int; // Δυναμική δέσμευση μνήμης
    if (!p) cout<<"Error allocating memory";

    /* Αναθέσεις τιμών */
    x = 1;
    *p = 5;
    /* Εκτυπώσεις */
    cout<<"x = "<<x<<" (address: "<<&x<<)"<<endl;
    cout<<"*p = "<<*p<<" (address: "<<p<<)"<<endl;

    delete p; // Αποδέσμευση μνήμης

    return 0;
}
```

2. Δυναμική Δέσμευση Μνήμης

4. Δυναμική Δέσμευση για Αντικείμενα

- Αντίστοιχα (αφού μία κλάση είναι τύπος δεδομένων), μπορεί να δεσμευτεί δυναμικά χώρος μνήμης για ένα αντικείμενο.
- Όταν χρησιμοποιούμε ένα αντικείμενο μέσω δείκτη,
 - Έχουμε πρόσβαση στα δημόσια μέλη-μεθόδους μέσω του τελεστή -> (βελάκι)
- Βλέπουμε και ένα παράδειγμα:

```
/* CPP3.dynamic_object.cpp Δυναμική Δέσμευση
μνήμης για αντικείμενο */
#include <iostream>
using namespace std;

class dummy {
public:
    int x;
};
```

```
int main()
{
    dummy *p = NULL; //Θα δείξει σε αντικείμενο
    p = new dummy; // Δέσμευση χώρου
    if (!p) cout<<"Error allocating memory";

    /* Αναθέσεις τιμών */
    p->x = 5;

    /* Εκτυπώσεις */
    cout<<"p->x = "<<p->x<<endl;

    delete p; // Αποδέσμευση μνήμης

    return 0;
}
```

2. Δυναμική Δέσμευση Μνήμης

4. Δυναμική Δέσμευση για Αντικείμενα

- Δεδομένου ενός δείκτη που δείχνει σε ένα αντικείμενο.
 - Με τον τελεστή *, έχουμε πρόσβαση στο ίδιο το αντικείμενο (dereferencing)
 - Και ισχύει ότι το (*p) είναι το αντικείμενο στο οποίο δείχνει ο δείκτης.
 - Συνεπώς έχουμε πρόσβαση στα δημόσια μέλη του (*p) με τον τελεστή . (τελεία)
- Βλέπουμε και ένα παράδειγμα:

```
/* CPP3.dereferencing.cpp */  
#include <iostream>  
using namespace std;  
  
class dummy {  
public:  
    int x;  
};
```

```
int main()  
{  
    dummy *p = NULL; //Θα δείξει σε αντικείμενο  
    p = new dummy; // Δέσμευση χώρου  
    if (!p) cout<<"Error allocating memory";  
  
    /* Αναθέσεις τιμών */  
    (*p).x = 5;  
  
    /* Εκτυπώσεις */  
    cout<<"(*p).x = "<<(*p).x<<endl;  
  
    delete p; // Αποδέσμευση μνήμης  
  
    return 0;  
}
```

2. Δυναμική Δέσμευση Μνήμης

4. Δυναμική Δέσμευση για Αντικείμενα

- Και ένα ακόμη παράδειγμα, όπου βάζουμε ένα δείκτη να δείχνει σε ένα ήδη υφιστάμενο αντικείμενο:
 - Για να τονιστεί η διαφορά στην πρόσβαση,
 - απευθείας από το αντικείμενο (τελεστής .)
 - μέσω δείκτη (τελεστής ->)
 - μέσω dereferencing (* και .)

```
/* CPP3.pointers_and_objects.cpp Δείκτης σε  
αντικείμενο */  
#include <iostream>  
using namespace std;  
  
class dummy {  
public:  
    int x;  
};
```

```
int main()  
{  
    dummy ob;  
    dummy *p = &ob;  
    if (!p) cout<<"Error allocating memory";  
  
    /* Αναθέσεις τιμών */  
    ob.x = 6;  
  
    /* Εκτυπώσεις */  
    cout<<ob.x<<" "<<p->x<<" "<<(*p).x<<endl;  
  
    return 0;  
}
```


2. Δυναμική Δέσμευση Μνήμης

5. Δυναμική Δέσμευση και κατασκευαστές

- Όταν δημιουργούμε ένα αντικείμενο με τον τελεστή new,
 - και εφόσον το αντικείμενο αυτό έχει κατασκευαστή
 - πρέπει να περάσουμε και τα ορίσματα στον κατασκευαστή
- **Ο κατασκευαστής καλείται όταν κάνουμε new**
- **Ο καταστροφέας καλείται όταν κάνουμε delete**

```
/*
CPP3.constructor_destru
ctor.cpp */
#include <iostream>
using namespace std;

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

```
int main()
{
    dummy *p = NULL;

    p = new dummy(5); //constructing
    if (!p) cout<<"Error allocating
                    memory";
    delete p; //destructing
    return 0;
}
```

```
dummy::dummy(int in_x)
{
    x = in_x;
    cout<<"Constructing...";
}

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


3. Κλάσεις που περιέχουν δείκτες

1. Παράδειγμα κλάσης που περιέχει δείκτη

- Ένας δείκτης είναι απλά μία μεταβλητή
 - Συνεπώς μπορεί να είναι μέλος σε κλάση
- Βλέπουμε ένα απλό παράδειγμα:
 - Στο οποίο δεσμεύουμε δυναμικά στον κατασκευαστή το χώρο για έναν ακέραιο
 - και τον αποδεσμεύουμε στον καταστροφέα.

```
/* CPP3.class_with_pointer */  
#include <iostream>  
using namespace std;  
  
class dummy {  
public:  
    dummy();  
    ~dummy();  
    void set_val(int in_val);  
    int get_val();  
private:  
    int *p_val;  
};
```

3. Κλάσεις που περιέχουν δείκτες

1. Παράδειγμα κλάσης που περιέχει δείκτη

```
int main()
{
    dummy ob;

    ob.set_val(3);

    cout<<endl<<ob.get_val()<<endl;

    return 0;
}

dummy::dummy()
{
    p_val = new int;
    if (!p_val) cout<<"Error allocating memory";

    cout<<"Constructing...";
}
```

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

void dummy::set_val(int in_val)
{
    *p_val = in_val;
}

int dummy::get_val()
{
    return *p_val;
}
```

3. Κλάσεις που περιέχουν δείκτες

2. ...και ένα πρόβλημα (χωρίς λύση για την ώρα)

- Ας δούμε και ένα πρόβλημα που μπορεί να προκύψει όταν μία κλάση περιέχει δείκτες που κάνουν δυναμική διαχείριση μνήμης
- Το πρόβλημα αυτό προκύπτει όταν:
 - Έχουμε μία κλάση που κάνει δυναμική δέσμευση μνήμης
 - Κάνουμε αντιγραφή του αντικειμένου (π.χ. ob1 = ob2)
- Μεταγλωττίστε και εκτελέστε το ακόλουθο πρόγραμμα:

```
/* CPP3.assignment_problem Δυναμική Δέσμευση και Τελεστής Ανάθεσης */  
#include <iostream>  
using namespace std;  
  
class dummy {  
public:  
    dummy();  
    ~dummy();  
    void set_val(int in_val);  
    int get_val();  
private:  
    int *p_val;  
};
```

3. Κλάσεις που περιέχουν δείκτες

2. ...και ένα πρόβλημα (χωρίς λύση για την ώρα)

```
int main()
{
    dummy ob1;
    ob1.set_val(3);
    dummy ob2;
    ob2 = ob1;

    cout<<ob1.get_val()<<endl;
    cout<<ob2.get_val()<<endl;

    return 0;
}

dummy::dummy()
{
    p_val = new int;
    if (!p_val) cout<<"Error allocating memory";

    cout<<"Constructing...";
}
```

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

void dummy::set_val(int in_val)
{
    *p_val = in_val;
}

int dummy::get_val()
{
    return *p_val;
}
```

- Σημαντικό! Στη C++ οι μεταβλητές μπορούν να δηλωθούν σε οποιοδήποτε μέρος του προγράμματος (αλλά θα το αποφεύγουμε...)



ΤΕΛΟΣ ΜΑΘΗΜΑΤΟΣ

ΚΛΑΣΕΙΣ ΚΑΙ ΔΕΙΚΤΕΣ

Γιώργος Διάκος - Full Stack Developer