Arrays – pointers

Arrays

Arrays

- Οι arrays χρησιμοποιούνται όταν θέλουμε να δηλώσουμε πολλές μεταβλητές του ΙΔΙΟΥ τύπου.
  - Π.χ. μια σειρά από θερμοκρασίες
- Με αυτόν τον τρόπο λύνουμε το πρόβλημα στο να δίνουμε διαφορετικό όνομα για κάθε μεταβλητή
- Οι τιμές που αποτελούν έναν πίνακα ονομάζονται στοιχεία του πίνακα
- Η C++ εγγυάται ότι οι τιμές αυτές θα τοποθετηθούν σε διαδοχικές θέσεις στην μνήμη.

Δήλωση ενός πίνακα

- Η δήλωση μιας μεταβλητής πίνακα γίνεται με τον παρακάτω τρόπο

base_type var_name[size]

- Όπου
  - **Base_type** είναι ο κοινός τύπος όλων των στοιχείων του πίνακα
  - **Var_name** είναι το όνομα της μεταβλητής που δηλώνουμε
  - **Size** είναι το μέγεθος του πίνακα (ο αριθμός των στοιχείων του)

```
int a[10];
string words[20];
```

Αρχικοποίηση πίνακα

- Το μέγεθος ενός πίνακα ΠΡΕΠΕΙ να είναι γνωστό κατά την στιγμή της δήλωσης. Επομένως σαν μέγεθος μπορούν να μπουν μόνο const
- **ΤΟ ΜΕΓΕΘΟΣ ΔΕΝ ΑΛΛΑΖΕΙ!**
- Μπορούμε να αρχικοποιήσουμε ολόκληρο ή μέρος ενός πίνακα (τα υπόλοιπα αρχικοποιούνται σε μηδέν)
- ΠΡΟΣΟΧΗ! **Δεν μπορούμε να κάνουμε ανάθεση σε πίνακα**!!!

```
const int x = 20;
int a[x];
int b[] = { 3,4,5 };
int c[5] = { 1,2,3 };
```

Πρόσβαση στα στοιχεία ενός πίνακα

- Η πρόσβαση στα στοιχεία ενός πίνακα γίνεται με την χρήση indexing
- ΠΡΟΣΟΧΗ!!!
- Στην C++ τα index ξεκινάν από τον αριθμό 0 και τελειώνουν στο size-1

```
int b[] = { 3,4,5 };

cout << b[0] << endl;
cout << b[2] << endl;
```

- ΠΡΟΣΟΧΗ! Ένα πολύ συχνό λάθος είναι η χρήση index που είναι εκτός ορίων!

Χρήση των στοιχείων ενός πίνακα

- Τα στοιχεία ενός πίνακα μπορούν να χρησιμοποιηθούν όπως ακριβώς και οι απλές μεταβλητές.
  - Ανάθεση
  - Χρήση σε εκφράσεις
  - Όρισμα σε συνάρτηση
- Το index μπορεί να είναι μια οποιαδήποτε έκφραση που επιστρέφει ακέραιο

```
b[0] = 55;
b[2] = 4 + b[1];
b[2] = pow(b[1], 2);
```

Πρόσβαση σε όλα τα στοιχεία (range for)

- Ένας τρόπος για την πρόσβαση σε όλα τα στοιχεία ενός array είναι με την χρήση for όπως είχαμε δει και στα string
- Αν θέλουμε να έχουμε μεταβολή των στοιχείων η πρόσβαση πρέπει να γίνεται με αναφορά

```
int start = 10;
int a[5];

for (auto &x : a)
    x = ++start;

for (auto x : a)
    cout << x << endl;
```

Πρόσβαση σε όλα τα στοιχεία (index)

- Ένας άλλος τρόπος για την πρόσβαση σε όλα τα στοιχεία του πίνακα είναι με την χρήση index και for

- Μπορούμε να βρούμε το πλήθος των στοιχείων και από το μέγεθος

```
int start = 10;
int a[5];

for (int i = 0; i < 5; i++)
{
    a[i] = ++start;
}

size = sizeof(a) / sizeof(a[0]);
```

Πίνακες και συναρτήσεις

Πέρασμα ολόκληρου πίνακα σαν όρισμα

- Πολλές φορές χρειάζεται να «περάσουμε» σαν όρισμα ολόκληρο τον πίνακα
- Στο function call γράφουμε απλώς το όνομα του πίνακα
- Η αντίστοιχη παράμετρος στην δήλωση της συνάρτησης μπορεί να γραφεί με τρεις τρόπους
  - Πίνακας με μέγεθος
  - Πίνακας με άδειο μέγεθος
  - Pointer

```
double calc_mean(int m[5]);
double calc_mean(int m[]);
double calc_mean(int *m);

int main()
{
    int a[] = { 4,5,6,7,8 };
    double mean = 0.0;

    mean = calc_mean(a);

    cout << "Mesos oros " << mean << endl;

}
```

Πέρασμα ολόκληρου πίνακα σαν όρισμα

- Όταν περνάμε ολόκληρο πίνακα σαν όρισμα έχουμε τις παρακάτω ιδιαιτερότητες
  - ΔΕΝ γνωρίζουμε το μέγεθος του πίνακα μέσα στην συνάρτηση
  - Στην ουσία «περνάμε» την διεύθυνση του πρώτου στοιχείου
  - Μπορούμε να μεταβάλουμε τον πίνακα σαν να τον είχαμε περάσει με αναφορά

```
void my_func(int m[]);

int main()
{
    int a[] = { 4,5,6,7,8 };

    my_func(a);

    cout << "a[2]= " << a[2] << endl;
}

void my_func(int m[]) {
    cout << sizeof(m) << endl;
    m[2] = 99;
}
```

Πέρασμα ολόκληρου πίνακα σαν όρισμα

- Αν θέλουμε να γνωρίζουμε και το μέγεθος του πίνακα μέσα στην συνάρτηση πρέπει να το περάσουμε σαν όρισμα
- Αν δεν θέλουμε να μεταβάλουμε τον πίνακα μέσα στην συνάρτηση τότε πρέπει να ορίζουμε την παράμετρο σαν const

```
void my_func(const int m[],int size);

int main()
{
    int a[] = { 4,5,6,7,8 };

    my_func(a,sizeof(a)/sizeof(int));

    cout << "a[2]= " << a[2] << endl;
}

void my_func(const int m[],int size) {
    cout << m[size - 1];
    m[0] = 99;
}
```

Αναφορές και πίνακες

- ΔΕΝ μπορούμε να έχουμε πίνακα με στοιχεία αναφορές μιας και οι αναφορές ΔΕΝ είναι αντικείμενα και δεν μπορούν να τοποθετηθούν διαδοχικά στην μνήμη
- Μπορούμε να δηλώσουμε αναφορά σε έναν πίνακα αρκεί να δηλώνουμε και το μέγεθος του (δεν μπορεί να το «βρει» από την αρχικοποίηση)

```
int m[] = { 1,2,3,4,5 };

int &a[5] = m;

int(&a)[] = m;

int (&a)[5] = m;
```

Χρήση range for μέσα σε συνάρτηση

- Όταν περνάμε έναν ολόκληρο πίνακα σαν όρισμα σε μία συνάρτηση στην ουσία δεν περνάμε πίνακα αλλά pointer (στο επόμενο)
- Επομένως ΔΕΝ μπορούμε να χρησιμοποιήσουμε range for
- Μπορούμε όμως αν περάσουμε αναφορά στον πίνακα

```
double calc_mean(int (&a)[5]);

int main()
{
    int m[] = { 1,2,3,4,5 };
    int (&a)[5] = m;
    double mo;

    mo = calc_mean(m);
    cout << mo << endl;
    return 0;
}

double calc_mean(int (&a)[5])
{
    int sum = 0;

    for (auto x : a) {
        sum += x;
    }
}
```

Πολυδιάστατοι πίνακες

Πίνακες πολλών διαστάσεων

- Μπορούμε να δηλώσουμε έναν πίνακα που να έχει παραπάνω από μία διαστάσεις
- Για παράδειγμα

int arr[20][30];

- Δηλώνει έναν πίνακα από ακεραίους με 20 γραμμές και 30 στήλες
- Στην πραγματικότητα είναι ένας πίνακας με 20 στοιχεία κάθε ένα από τα οποία είναι ένας πίνακας των 30 ακεραίων

Αρχικοποίηση πολυδιάστατων πινάκων

- Μπορούμε να αρχικοποιήσουμε πίνακες πολλών διαστάσεων με άγκιστρα
- ΠΡΟΣΟΧΗ μόνο η πρώτη διάσταση μπορεί να παραληφθεί
- Σε περίπτωση που παραλειφθούν στοιχεία συμπληρώνονται με μηδέν

```
int m[2][3] = {
    {1,2,3},
    {4,5,6} };

int k[][3] = {
    {4,5,6},
    {3,7,8}
};

int l[][3] = {
    {6},
    {5}
};
```

Αναφορά σε στοιχεία πολυδιάστατου πίνακα

- Για να αναφερθούμε σε στοιχείο ενός πολυδιάστατου πίνακα πρέπει να χρησιμοποιήσουμε ένα index για κάθε διάσταση
- Σε περίπτωση που παραλείψουμε μία διάσταση τότε έχουμε πρόσβαση σε έναν υπο-πίνακα

```
int m[2][3] = {
    {1,2,3},
    {4,5,6} };

cout << m[1][1] << endl;
// aplos akeraios

int(&x)[3] = m[0];
// pinakas me 3 akeraious

cout << x[1] << endl;
```

Πρόσβαση σε όλα τα στοιχεία ενός δισδιάστατου πίνακα

- Ο «κλασικός» τρόπος πρόσβασης σε όλα τα στοιχεία ενός πολυδιάστατου πίνακα είναι με index και χρήση εμφωλευμένων for

```
int m[2][3] = {
    {1,2,3},
    {4,5,6} };

for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 3; ++j)
        cout << m[i][j] << " ";
    cout << endl;
}
```

Πρόσβαση σε όλα τα στοιχεία ενός δισδιάστατου πίνακα

- Μπορεί να γίνει πρόσβαση σε όλα τα στοιχεία και με την χρήση εμφωλευμένων range for
- ΠΡΟΣΟΧΗ! Εκτός από την τελευταία οι προηγούμενες πρέπει να είναι με αναφορά

```
int m[2][3] = {
    {1,2,3},
    {4,5,6} };

for (auto &row : m) {
    for (auto elem : row)
        cout << elem << " ";
    cout << endl;
}
```

Πίνακας πολλών διαστάσεων σαν όρισμα

- Σε περίπτωση που θέλουμε να περάσουμε πολυδιάστατο πίνακα σαν όρισμα μπορούμε να μην περιγράψουμε ΜΟΝΟ το μέγεθος της πρώτης διάστασης
- Όλες οι επόμενες διαστάσεις πρέπει να περιγράφονται

```
int main()
{
    int a[2][4] = {
        {1,2,3,4}, {5,6,7,8}
    };

    cout << "MO="<<calc_mean(a)<<endl;
}

double calc_mean(int m[][4])
{
    double mo = 0.0;
    int i = 0, j = 0;
    for (i = 0; i < 2; ++i)
        for (j = 0; j < 4; ++j)
            mo += m[i][j];

    return mo / (i*j);
}
```

Pointers

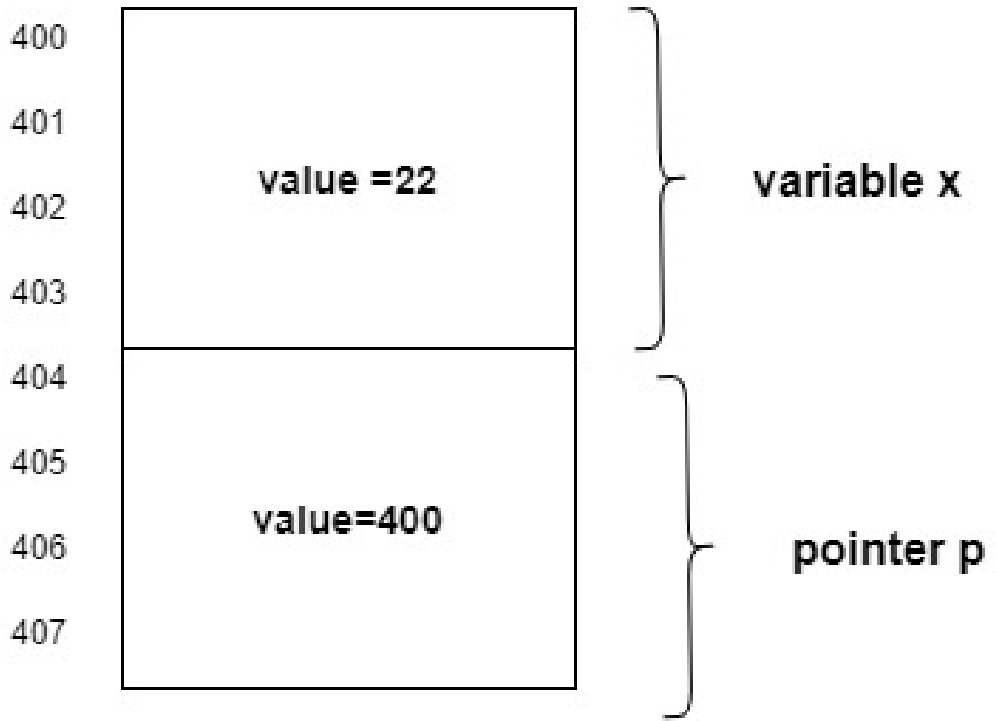
Pointer

- Pointer είναι μια μεταβλητή που περιέχει σαν τιμή μια διεύθυνση της μνήμης
- Στην ουσία μπορούμε να έχουμε πρόσβαση σε ένα αντικείμενο στην μνήμη με τρεις τρόπους
  - Με το όνομα της μεταβλητής που το έχει σαν τιμή
  - Με μία αναφορά σε αυτή την μεταβλητή
  - Με έναν pointer που έχει τιμή την διεύθυνση στην οποία είναι αποθηκευμένο αυτό το αντικείμενο

Pointers

- Η διεύθυνση μνήμης μιας μεταβλητής που «απλώνεται» σε παραπάνω από ένα byte είναι η διεύθυνση του αρχικού byte
- Ο compiler πρέπει να γνωρίζει σε τι τύπο μεταβλητής «δείχνει» ένας Pointer για να συμπεριλαμβάνει και τα αντίστοιχα επόμενα byte

```
int x = 22;

int *p = &x;
```



Δήλωση pointer

- Η δήλωση ενός pointer γίνεται με την χρήση του χαρακτήρα * πριν από το όνομα της μεταβλητής
- Μπορούμε σε μία δήλωση να έχουμε και κανονικές μεταβλητές και μεταβλητές τύπου pointer
- Ο τελεστής & επιστρέφει την θέση μνήμης μιας μεταβλητής και μπορεί να χρησιμοποιηθεί στην ανάθεση τιμής σε μια μεταβλητή τύπου pointer
- ΠΡΟΣΟΧΗ! Αν δεν αρχικοποιηθεί έχει ακαθόριστη τιμή

```
int x, *p;

x = 12;
p = &x;
```

Χρήση του pointer

- Από την στιγμή που έχει δημιουργηθεί ένας pointer (π.χ. int *p) μπορούμε να τον χρησιμοποιήσουμε με δύο τρόπους
  - Σαν κανονική μεταβλητή (δηλαδή την διεύθυνση που δείχνει)
  - Σαν αναφορά στην θέση μνήμης στην οποία δείχνει (dereference με την χρήση του χαρακτήρα *)

```
int x, *p;

x = 12;
p = &x;

cout << dec <<(long long) p << endl;
cout << *p << endl;

*p = 33;    // allazei to x
```

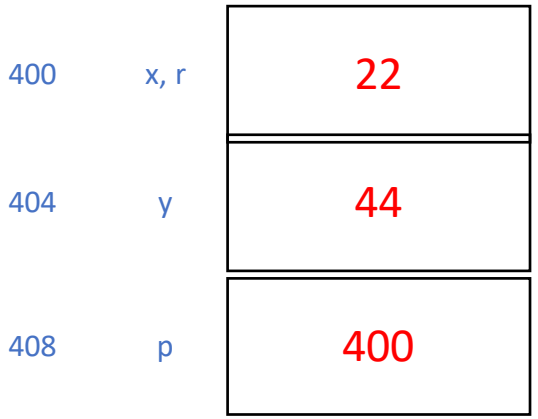
Τελεστές [], & ,*

- έχουν διαφορετική λογική όταν χρησιμοποιούνται σε μία εντολή δήλωσης και σε μία έκφραση
- Δήλωση
  - [] μέγεθος πίνακα
  - & δήλωση reference
  - * δήλωση pointer
- Expression
  - [] δείκτης (ποιο στοιχείο)
  - & η θέση μνήμης ενός αντικειμένου
  - * dereference (για pointer)

```
int x = 12;
int a[10] = { 0 };
int &b = x;
int *c;

a[3] = 44;
c = &x;
*c = 55;
```

Διαφορές pointer και reference

- Και με τους δύο τρόπους έχουμε «έμμεση» πρόσβαση σε άλλη μνήμη



Pointer	Reference
Δημιουργείται αντικείμενο	ΔΕΝ δημιουργείται αντικείμενο
ΔΕΝ θέλει αρχικοποίηση	ΑΠΑΡΑΙΤΗΤΗ αρχικοποίηση
Μπορεί να δείξει σε καινούργια διεύθυνση μνήμης	Δείχνει ΠΑΝΤΑ στην ίδια θέση μνήμης

Τιμές Pointer

- Οι τιμές που μπορεί να πάρει ένας pointer είναι
  - Να δείχνει σε ένα αντικείμενο
  - Να έχει nullptr
  - Να δείχνει σε μια θέση ΜΕΤΑ από ένα αντικείμενο
- Στις δύο τελευταίες περιπτώσεις είναι περιορισμένες οι ενέργειες που μπορούμε να κάνουμε με αυτό

```
int a[] = {3,5,7,9,6};

int *cur = a;
int *end = cur + 5;

while (cur != end) {
    cout << *(cur++)<<endl;
}
```

Παράδειγμα pointers2

Τελεστές με Pointers

Ανάθεση σε pointer

- Οι pointers έχουν σαν τιμή διευθύνσεις στην μνήμη. Αυτές μπορεί να είναι ακέραιοι αλλά δεν μπορούν να γίνουν όλες οι πράξεις
- Ανάθεση
- Μπορεί να γίνει η ανάθεση της διεύθυνσης μνήμης μιας άλλης μεταβλητής
- ΔΕΝ μπορεί να ανατεθεί κατευθείαν ένας ακέραιος
- Μπορεί να ανατεθεί η ειδική τιμή nullptr (δεν «δείχνει» πουθενά)

```
int x = 12;
int *p;

p = &x;
*p = 23;
p = nullptr;

p = 322; // error
```

Ο τελεστής new

- Ένας άλλος ιδιαίτερα χρήσιμος τελεστής για τους Pointers είναι ο τελεστής new
- Με αυτόν τον τελεστή ο compiler «δεσμεύει» δυναμικά χώρο
- Ο χώρος αυτός υπάρχει σε μια περιοχή που λέγεται heap και είναι ανεξάρτητος από την μνήμη stack που χρησιμοποιείται για τις τοπικές μεταβλητές
- Απελευθερώνεται με τον τελεστή delete

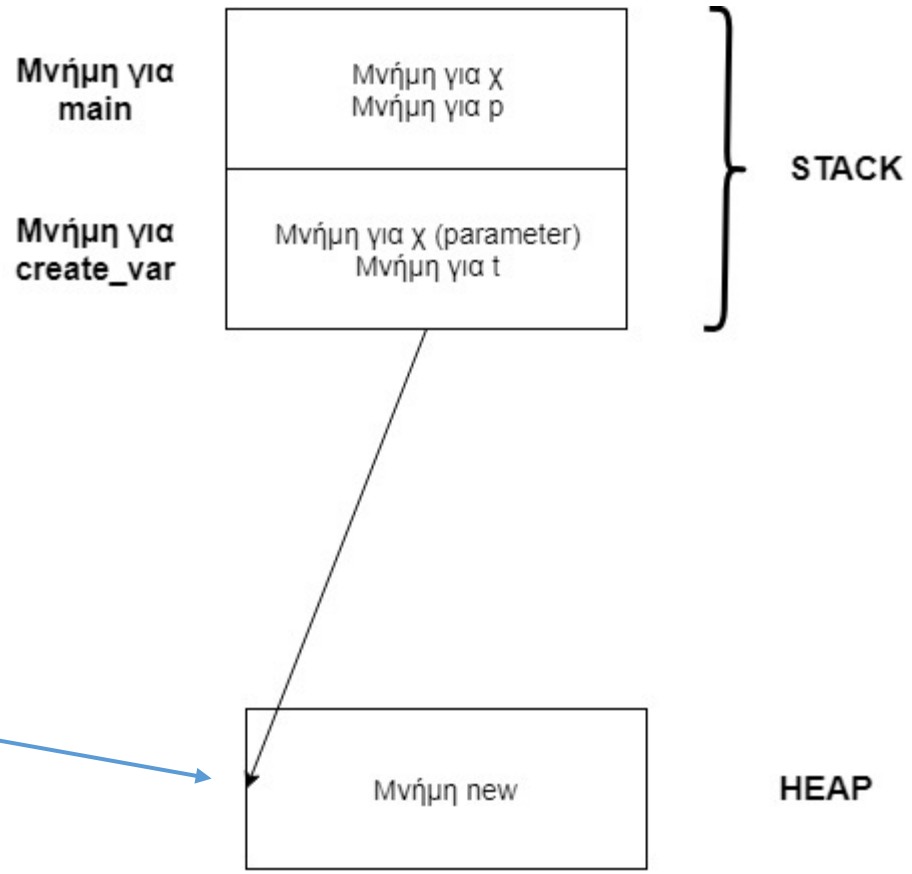
```
int x = 12;
int y = 33;
int *p;

p = new int;
*p = 22;

cout << dec << (long long) &x << endl;
cout << dec << (long long)&y << endl;
cout << dec << (long long)p << endl;
```

Παράδειγμα new1

Παράδειγμα δυναμικής μεταβλητής

```
int main()
{
    int x = 5;
    int *p = nullptr;

    p = create_var(22);
    cout << *p;
    delete p;
}

int *create_var(int x) {
    int *t = nullptr;

    t = new int;
    *t = x;

    return t;
}
```

t «δείχνει» στην καινούργια μνήμη (π.χ. 2000)
Η θέση αυτή επιστρέφεται από την συνάρτηση



Παράδειγμα dynvariable

Παράδειγμα δυναμικής μεταβλητής

```
int main()
{
    int *point;
    point = create_var(44);
    cout << "Dynamic var="<<*point << endl;
    delete point;
}
```

```
Microsoft Visual Studio Debug Console
creating new dynamic variable
Dynamic var=44

C:\Users\groban\source\repos\th7\Debug\th7
Press any key to close this window . . .
```

```
int *create_var(int x)
{
    int *p;
    cout << "creating new dynamic variable\n";
    p = new int;
    *p = x;
    return p;
}
```

Εδώ ο pointer p παίρνει σαν τιμή την αρχή της διεύθυνσης που «δέσμευσε» ο compiler

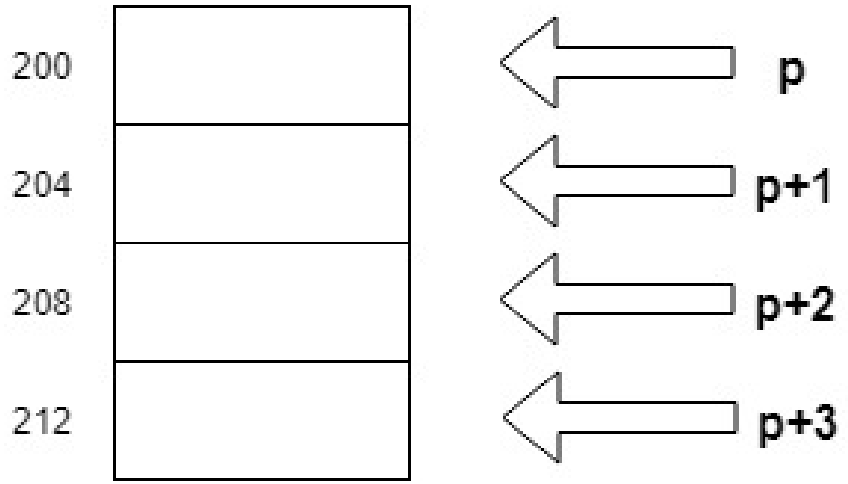
Δυναμικοί πίνακες

- Με την χρήση του τελεστή new μπορούμε να «δεσμεύσουμε» μνήμη για παραπάνω από μία μεταβλητή
- Με αυτόν τον τρόπο δημιουργούμε ***«δυναμικούς» πίνακες***
- Η πρόσβαση στα στοιχεία μπορεί να γίνει σαν να είναι κανονικός πίνακας
- **(παράδειγμα dynarray)**

```
int *p, size;

cout << "Dose to megethos tou pinaka: ";
cin >> size;
p = new int[size];

for (int i = 0; i < size; ++i)
    p[i] = 0;

for (int i = 0; i < size; ++i)
    cout << p[i] << endl;
return 0;
```

Πρόσθεση σε pointer

- Μπορούμε να προσθέσουμε σε pointer έναν ακέραιο. π.χ. το 2
- Αυτό που σημαίνει είναι, σε περίπτωση που ο Pointer «δείχνει» σε πίνακα, ο pointer να «δείξει» σε 2 στοιχεία παρακάτω..
- Το να προσθέσουμε δύο pointers δεν έχει κάποια ιδιαίτερη σημασία



Αν προσθέσω 1 σε έναν pointer ΔΕΝ σημαίνει ότι θα «δείξει» στο επόμενο byte ΑΛΛΑ θα «δείξει» στο επόμενο στοιχείο από εκείνο που ήδη «δείχνει»

Πρόσθεση σε pointer

- ΠΡΟΣΟΧΗ στην προτεραιότητα των τελεστών!
- Άλλο το νόημα της έκφρασης

p+1

- και άλλο της

(p+1)

- Επίσης προσοχή! Ο τελεστής ++ έχει μεγαλύτερη προτεραιότητα

p++

```
int a[4] = {10,20,30,40};

int *p;

p = a;
// o deiktis p stin arxi toy a

cout << *p + 1 << endl;
// typonei 11

cout << *(p + 1) << endl;
// typonei 20
```

Πρόσθεση σε pointer

```
int *p = nullptr;

p = new int[3];
//esto p = 200
*p = 10; // int stin thesi 200 (200-203)
*(p + 1) = 11; //int stin thesi 204 (204-207)
*(p + 2) = 12; //int stin thesi 208 (208-211)

for (int i = 0; i < 3; ++i)
    cout << *(p + i) << endl;

delete []p;
```

Παράδειγμα dynarray2

Αποδέσμευση Δυναμικού Πίνακα

Αφαίρεση pointer

- Η αφαίρεση ακεραίου από pointer έχει το νόημα ότι βρίσκουμε προηγούμενα στοιχεία ενός πίνακα
- Αν αφαιρέσουμε δύο pointers που «δείχνουν» στον ίδιο πίνακα τότε βρίσκουμε τον αριθμό των στοιχείων που διαφέρουν
- **(παράδειγμα pointerarithm)**

```
int a[10] = {};
for (int i = 0; i < 10; ++i)
    a[i] = i;

int *p = &a[6], *t=&a[3];
// p deixnei sto 7o stoixeio

*(p - 2) = 33;
// to 5o stoixeio einai 33;

int diff;
diff = p - t;  //pairnei tin timi 3
```

Σύγκριση Pointer

- Η σύγκριση μεταξύ Pointer έχει νόημα όταν «δείχνουν» σε αντικείμενα ίδιου τύπου
- Η ισότητα / ανισότητα ελέγχει αν «δείχνουν» στο ίδιο αντικείμενο
- Το μεγαλύτερο / μικρότερο αν δείχνουν σε προηγούμενο / επόμενο στοιχείο πίνακα

```
int a[4] = {10,20,30,40};

int *p, *t;

p = a;
t = &a[0];

cout << (p == t) << endl;
//true 1

p = &a[3];

cout << (p < t) << endl;
//false 0
```

Pointers και συναρτήσεις/ μέθοδοι

Pointer σαν όρισμα

- Μία από τις παραμέτρους μίας συνάρτησης μπορεί να είναι Pointer
- Στην κλήση της συνάρτησης θα πρέπει να δοθεί pointer που δείχνει στον ίδιο τύπο
- Σαν αποτέλεσμα είναι η συνάρτηση που καλέστηκε να μπορεί να μεταβάλει μνήμη που ανήκει στην συνάρτηση που την κάλεσε
- (**παράδειγμα pointerfunc**)

```
int main()
{
    int x = 22;

    cout << "Prin tin synartish " << x << endl;
    my_func(&x);
    cout << "Meta tin synartish " << x << endl;
}

void my_func(int *t) {
    *t = 33;
}
```

Συνάρτηση που επιστρέφει pointer

- Μια συνάρτηση μπορεί να επιστρέφει pointer
- Με αυτό τον τρόπο μπορεί η συνάρτηση που την κάλεσε έχει πρόσβαση σε συγκεκριμένη θέση μνήμης που:
  - Δημιούργησε η συνάρτηση που καλέστηκε (new)
  - Συγκεκριμένη θέση σε πίνακα (λίστα)
  - (**παράδειγμα pointerreturn**)

```
int main() {
    int a[] = { 11,33,44,55,66 };
    int *r;

    r = find(a, &a[4], 44);
    cout << "44 mnimi " << (long long)r << endl;
    r = find(a, &a[4], 55);
    cout << "55 mnimi " << (long long)r << endl;
}

int *find(int *start, int *end, int value) {
    for (int *t = start; t != end + 1; ++t)
        if (*t == value)
            return t;
    return nullptr;
}
```

Pointers και αντικείμενα

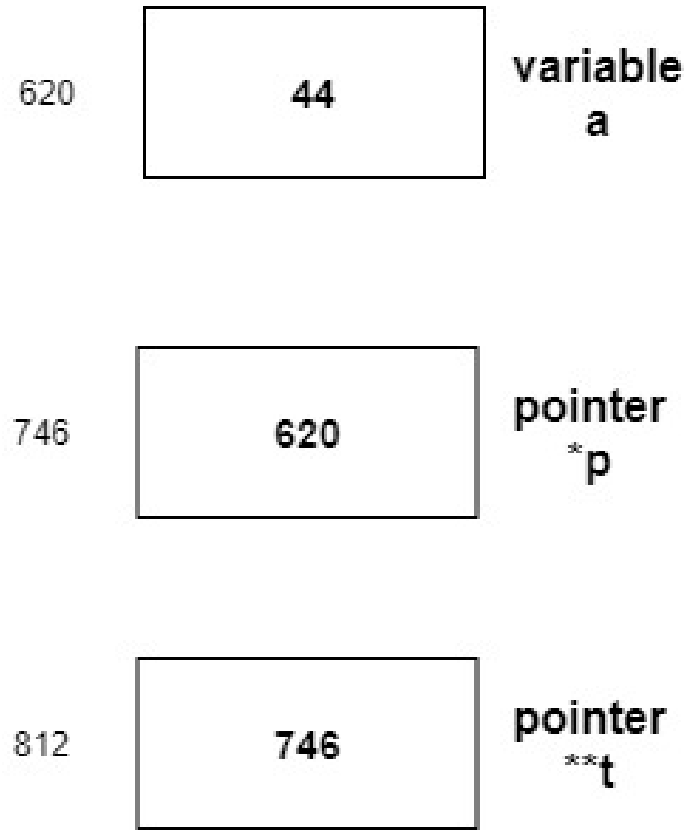
- Μπορούμε να έχουμε pointer σε οποιαδήποτε τύπο ακόμα και σε κλάσεις
- Η πρόσβαση στα στοιχεία ενός αντικειμένου που «δείχνει» ένας pointer μπορεί να γίνει με τον τελεστή «βέλος»
- (**παράδειγμα pointerobj**)

```
vector<int> ivec = { 3,4,6,7 };
vector<int> *pvec;

pvec = &ivec;

cout << pvec->size() << endl;;
cout << (*pvec).size() << endl;;

pvec->push_back(33);
pvec->clear();
```

Pointers σε pointers

- Ο pointer είναι ένα αντικείμενο στην μνήμη
- Επομένως μπορούμε να ορίσουμε έναν καινούργιο pointer που θα «δείχνει» σε αυτό
- Μπορούμε να έχουμε διάφορα επίπεδα πρόσβασης

```
int a = 44;
int *p = &a;
int **t = &p;
```



Όρισμα πίνακας δύο διαστάσεων με Pointer

- Μπορούμε να εκμεταλλευτούμε το γεγονός ότι η C++ αποθηκεύει διαδοχικά (ανά γραμμή) τα στοιχεία ενός πίνακα και με αυτό τον τρόπο να τα προσπελάσουμε με την χρήση απλού pointer

```
int main()
{
    int a[][3] = { {3,5,7},
                   {8,9,7} };

    int sum;
    sum = calc_sum((int *)a, 2, 3);
    cout << sum << endl;
}

int calc_sum(int *x, int rsize, int csize) {
    int xsum = 0;

    for (int i = 0; i < rsize; ++i)
        for (int j = 0; j < csize; ++j)
            xsum+= *(x + i * rsize + j);

    return xsum;
}
```