Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

C++

ΕΓΧΕΙΡΙΔΙΟ

και

ΑΣΚΗΣΕΙΣ

ΠΑΣΧΑΛΗΣ ΡΑΠΤΗΣ

Θεσσαλονίκη 1998

C++

Η C++ προερχεται απο την C (η C ειναι υποσυνολο της C++) και εχει επεκταθει ωστε να υποστηριζει αντικειμενοστρεφη προγραμματισμο.
Παραδειγμα προγραμματος στην **C++**

```
#include <iostream.h>

main()
{
    cout << "Hello C++" << endl;

    return(0);
}
```

Ειδικοι χαρακτηρες

\n	Νεα γραμμη (new line). Το στιγμα (cursor) τοποθετειται στη αρχη της επομενης γραμμης.
\t	Στηλοθετης (tab): τυπωνει 8 κενους (spaces) χαρακτηρες.
\b	Το στιγμα τοποθετειται μια στηλη πισω (backspace).
\r	Το στιγμα (cursor) τοποθετειται στη αρχη της ιδιας γραμμης (carriage return)
\f	Το στιγμα (cursor) τοποθετειται στη ιδια στηλη της επομενης γραμμης (Line feed).
****	Αναποδη καθετος (back slash)
\"	Διπλα εισαγωγικα (double quotes)
\0	Μηδενικος χαρακτηρας (null)
\a	Προειδοποιητικος ηχος (bell='\07')

-- Παραδειγμα

```
#include <iostream.h>
int main()
{
    cout << "Borland" << '\n' << "Microsoft\n";
    cout << "Ειπε \"Γεια χαρα\" ειρωνικα." << endl;
    cout << "Dos διαδρομη: a:\\drivers\\W95" << endl;
    cout << 'A' << '\t' << 'B' << '\n';
    return(0);
}
```

Αποτελεσμα

```
Borland
Microsoft
Ειπε "Γεια χαρα" ειρωνικα.
Dos διαδρομη: a:\drivers\W95
A    B
```

Τυποι Μεταβλητων

Τυποι μεταβλητων της C/C++ ειναι:

short	Μικρος ακεραιος
int	Ακεραιος
long	Μεγαλος ακεραιος
float	Πραγματικος
double	Πραγματικος διπλης ακριβειας
char	Χαρακτηρας

α) int a;	Μεταβλητη a τυπου int
β) long b, d, i;	Μεταβλητες b, d, i τυπου long
γ) char c, ch;	Μεταβλητες c και ch τυπου char

Υπαρχει και ο προσδιοριστης **unsigned** για **int**, **long**, **short**, και **char** τυπους μεταβλητων. Οι μεταβλητες που οριζονται unsigned ειναι μεταβλητες χωρις προσημο και παντα φερουν θετικους αριθμους.

α) unsigned int;	Δηλωση ενος ακεραιου χωρις προσημο
β) unsigned char;	Δηλωση ενος χαρακτηρα χωρις προσημο
γ) unsigned long;	Δηλωση ενος μεγαλου ακεραιου χωρις προσημο

Εαν σε μια μηχανη ο int ειναι 2 bytes τοτε ο **unsigned int** μπορει να λαβει τιμες απο 0 εως 2^{16}-1 (2 bytes = 16 bits)

Ο **unsigned char** που καταλαμβανει χωρο ενα byte, μπορει να λαβει τιμες 0 - 255 (2^8)

Τελεστης =

Ο τελεστης **=** απονεμει τιμη σε μια μεταβλητη (assignment operator). Στα αριστερα του τελεστη βρισκεται παντα μια μεταβλητη. Στα δεξια μπορει να ειναι μια σταθερα, ή μια μεταβλητη ή μια παρασταση.

μεταβλητη = σταθερα ή μεταβλητη ή παρασταση

```
α) i = 100;
β) k = i;
γ) j = 5*k + 1;
```

Σταθερες

Οι σταθερες (constants) οριζονται με δυο τροπους:

A) Ορισμος σταθερων με **#define**

```
#define  MAX  100              // Ακεραιος
#define  PI     3.14           // Πραγματικος
#define  YES   'Y'             // Χαρακτηρας
#define  MACH  "DEC Alpha"     // String
#define  MEN   "Choose \t one value \n"  // String
```

Οι σταθερες που οριζονται με #define δεν καταλαμβανουν χωρο στην μνημη. Τα ονοματα των σταθερων μεσα στο προγραμμα αντικαθισταται με τις τιμες τους κατα την διαρκεια την προεπεξεργασιας της μεταφρασης.

```
#define  MAX  100        // Ακεραιος
:
int kostos, timi;
:
kostos = timi * MAX;
```

Ο παραπανω κωδικας κατα την προεπεργασια μετατρεπεται σε:

```
kostos = timi * 100;
```

Β) Δηλωση σταθερων με τον προσδιοριστη **const**

Ο προσδιοριστης **const** οριζει σταθερες ολων των τυπων δεδομενων. Κατα την δηλωση διδεται και η τιμη της σταθερας.

α) `const int max = 100;`

β) `const char mi = 'M';`

γ) `const float pi = 3.1415;`

δ) `const char *txt = "Simple man";`

Αρχικες τιμες σε μεταβλητες

Α) Αρχικες τιμες σε ακεραιο **short**, **int**, **long**:

`j = 1234;`	Δεκαδικος αριθμος
`j = 0123;`	Το 0 (μηδεν) στην αρχη μιας σταθερας σημαινει οτι ο αριθμος ειναι γραμμενος στο οκταδικο (octal) συστημα μετρησης.
`h = 0x1F;`	Το 0x ή 0X στην αρχη μιας σταθερας σημαινει οτι ο αριθμος ειναι γραμμενος στο δεκαεξαδικο hexadeciamal) συστημα μετρησης.
`i = 345L;`	Το L ή l στο τελος μιας σταθερας σημαινει οτι ο αριθμος ειναι long.
`m = 234U;`	Το U ή u στο τελος μιας σταθερας σημαινει οτι ο αριθμος ειναι χωρις προσημο.
`j = 'A';`	Το j παιρνει την τιμη 65 ('A' = 65 στον πινακα των ASCII
`k = '\x41';`	

Β) Αρχικες τιμες σε πραγματικο **float**, **double**:

```
j = 123.45;
j = 0.1e-2;  ή  j = 0.1E-2;
```

`j = 4.0F;` Το F ή f στο τελος μιας πραγματικης σταθερας σημαινει οτι ο αριθμος ειναι πραγματικος.

Γ) Αρχικες τιμες σε χαρακτηρα **char**:

`c = 'N';`	
`ch = 65;`	Ειναι ισοδυναμο με το ch = 'A'. Το γραμμα Α στον πινακα ASCII εχει τιμη 65.
`c = '\012';`	Το \ σημαινει ακολουθει χαρακτηρας, το 0 οτι ο αριθμος ειναι οκταδικος. Η μεταβλητη c παιρνει τον χαρακτηρα που αντιστοιχει στον αριθμο 012 (octal) = 10 (decimal).
`x = '\xA';`	Δεκαεξαδικο συστημα
`x = '\n';`	Νεα γραμμη
`x = '\07';`	Beep, bell

`x = '\0';` Χαρακτηρας null

Απονομη αρχικων τιμων δυναμικα

```
int x = 3*5;       // x = 15;
int y = 2*x + 3;   // y = 33;
int z = strlen("ocean sea");   // z = 9;
```

cin και cout

Η **cin** ειναι η καθιερωμενη εισοδος (πληκτρολογιο). Διαβαζει δεδομενα απο το buffer εισοδου και το κατευθυνει με τον τελεστη **>>** σε μεταβλητες του προγραμματος.

```
α) char ch;
   cin >> ch;  // Διαβαζει εναν χαρακτηρα απο το πληκτρολογιο και το
               // τοποθετει στο ch.
β) int i; char s[30];
   cin >> i >> s;  // Διαβαζει εναν ακεραιο και ενα string.
```

Η **cout** ειναι η καθιερωμενη εξοδος (οθονη). Τα δεδομενα κατευθυνονται με τον τελεστη **<<** στη οθονη.

```
α) char ch='R';
   cout << ch;  // Ο χαρακτηρας 'R' εμφανιζεται στη οθονη
β) int i=30; char *s = "life"
   cout << "For " << i << " years of " << s << '\n';
```

Θα εμφανισθει: **For 30 years of life**

Χειριστες μορφοποιησης

Οι χειριστες (manipulators) μορφοποιησης εισοδου/εξοδου οριζονται στα αρχεια επικεφαλιδων **iostream.h** και **iomanip.h**

endl Γραφει το '\n' (νεα γραμμη) [Εξοδο]

```
cout << "Windy Town" << endl;
```

ιδιο με το

```
cout << "Windy Town" << '\n';
```

ends Τερματιζει ενα string (ισοδυναμο με '\0') [Εξοδο]

setw(*ακεραιος*) Καθοριζει το ευρος εμφανισης (set width) [Εξοδο]

```
cout << setw(5) << 123;
```

Τυπωνει `'  123'` με δυο κενα στην αρχη.

setfill(*χαρακτηρας*) [Εξοδο]

```
cout << setfill('*') << setw(5) << 123;
```

Τυπωνει `'**123'` με δυο `'*'` στην αρχη.

setbase(*ακεραιος*)

Αλλαζει την βαση για την εμφανιση ακεραιων αριθμων. Οπου *ακεραιος* ειναι ενα απο: 10 (decimal - δεκαδικο συστημα μετρησης), 8 (octal-οκταδικο), 16 (hexadecimal-δεκαεξαδικο) [Εξοδο]

```
cout << setbase(16) << 127;  // Εμφανιζει 7F
```

dec Ισοδυναμο με το **setbase(10)** [Εισοδο/Εξοδο]
oct Ισοδυναμο με το **setbase(8)** [Εισοδο/Εξοδο]
hex Ισοδυναμο με το **setbase(16)** [Εισοδο/Εξοδο]

```
i = 127;
cout << hex << i << " " << dec << i << " " << oct << i;
```

Εμφανιζει: **7F 127 177**

```
int i, j, k;
cin >> hex >> i >> oct >> j >> dec >> k;
```

setprecision(*ακεραιος*)

Καθοριζει το πληθος των ψηφιων στην εμφανιση πραγματικων αριθμων. Εξ' ορισμου ειναι 6 θεσεις ακριβειας [Εξοδο]

```
cout << 123.456789;  // Εμφανιζει: 123.456
cout << setprecision(3) << 24.567; // Εμφανιζει: 24.5
```

Εαν οι θεσεις ειναι λιγοτερες απο τον αριθμο τοτε ο αριθμος εμφανιζεται σε εκθετικη μορφη.

```
cout << setprecision(2) << 1234.5; // Εμφανιζει: 1.2e+02
```

flush Σωζει τα δεδομενα απο την προσωρινη μνημη στον σκληρο δισκο [Εξοδο]

```
cout << "East 17";
cout << flush;
```

setiosflags(long *f*)

Ενεργοποιει τις σημαιες που οριζονται στο ***f*** [Εισοδο/Εξοδο]

resetiosflags(long *f*)

Απενεργοποιει τις σημαιες που οριζονται στο ***f*** [Εισοδο/Εξοδο]

```
double k = 590;
cout << setiosflags(ios::uppercase | ios::scientific);
cout << k << endl;
```

Θα εμφανισθει: **5.900000E+02**

```
cout << resetiosflags(ios::uppercase) << k << endl;
```

Θα εμφανισθει: **5.900000e+02**

ws Προσπερνα τα προπορευομενα κενα διαστηματα [Εισοδο]

Σχολια

Υπαρχουν δυο τροποι για να γραφονται τα σχολια (comments):

1ος τροπος.

```
// Μετα τις δυο πλαγιες καθετους και μεχρι το τελος της γραμμης γραφονται
// τα σχολια
int age;    // Μεταβλητη ηλικιας
```

2ος τροπος (C like).

```
/*
  Το σχολιο μπορει να καταλαμβανει μια ή και περισσοτερες
  γραμμες αρχιζει με καθετο / και * και τελειωνει με *
  και καθετο /
*/
int grades;  /* μεταβλητη βαθμολογιας */
```

Αποφευγετε να εσωκλειετε σχολια μεσα σε σχολια:

```
i = 15;   /* αρχικη /* τιμη */ 15 Λαθος */
```

Τελεστης &

Ο τελεστης **&** διπλα σε μια μεταβλητη δινει την διευθυνση της μεταβλητης στην μνημη. πχ. &i, &fahr. Η διευθυνση ειναι παντα ενας αριθμος χωρις προσημο (unsigned)

```
#include <iostream.h>
int main()
{
    int i;
    long j;
    // Οι διευθυνσεις θα εμφανισθουν σε δεκαεξαδικη (hex) μορφη
    cout << "Διευθυνση του i ειναι " << &i << endl;
    cout << "Διευθυνση του j ειναι " << &j << endl;
    // Σε περιπτωση που με την cout δεν εμφανισθει τιποτε δοκιμασετε την printf
    // printf("Διευθυνση του i ειναι %x \n", &i);
    // printf("Διευθυνση του j ειναι %X \n", &j);
    return(0);
}
// p6.cpp //
```

Αποτελεσμα

```
Διευθυνση του i ειναι 0x8f7ffff4
Διευθυνση του j ειναι 0x8f7fffec
```

Το προγραμμα αυτο θα δωσει διαφορετικα αποτελεσματα σε διαφορετικους υπολογιστες και σε διαφορετικους μεταφραστες.

Τελεστης ::

Ο τελεστης :: (scope operator) δηλωνει την εμβελεια μιας μεταβλητης.

```
#include <iostream.h>
int i = 3;        // global i
main()
{
    int i = 7;    // Τοπικο i
    cout << "Η τιμη του τοπικου i ειναι " << i   << endl;
    cout << "Η τιμη του global  i ειναι " << ::i << endl;
    return(0);
}
```

Τελεστης sizeof

Ο τελεστης **sizeof** δινει ενα ακεραιο ο οποιος αντιπροσωπευει το μεγεθος σε bytes ενος τυπου δεδομενων (μιας μεταβλητης, μιας δομης, σταθερας, αντικειμενου κ.ο.κ.)

Το μεγεθος που καταλαμβανει ενας τυπος εξαρταται απο το Η/Υ (16, 32 ή 64 bits επεξεργαστης) και απο τον μεταφραστη (compiler 16, 32 bits).

```
α) cout << sizeof(int);  // Ευρεση μηκους (σε bytes) του int
β) int i;
   cout << sizeof(i);    // Ευρεση μηκους (σε bytes) του i
γ) cout << sizeof(123);  /* Ευρεση μηκους (σε bytes) του ακεραιου
                            αριθμου 123 */
```

-- Παραδειγμα: Ευρεση αριθμου κελλιων ενος πινακα ακεραιων.

```
#include <iostream.h>
int main()
{
    int i,j,k;
    int ar[] = { 1, 2, 3, 4, 5 };  // Πινακας ar.
    i = sizeof(ar);    // i = τα bytes πιανει ο ar στην μνημη
    j = sizeof(ar[0]); // j = τα bytes πιανει ενα κελι του ar
                       // στην μνημη.
    k = i / j;         // k = το πληθος των κελλιων του ar.
    cout << "Ο πινακας ar εχει " << k << " κελια." << endl;
    return(0);
}
// p7.cpp //
```

Τελεστης ~

Το τελεστης ~ διδει το συμπληρωμα του ενος (ΣΤ1) ενος αριθμου ή μεταβλητης. Ολα τα bits ενος αριθμου αλλαζουν, εαν ειναι 0 γινονται 1 και εαν ειναι 1 γινονται 0.

```
α) int i;   // Για int μηκους 2 bytes i = ~0;
   i = ~0;  // το ΣΤ1 του 0000 ειναι ffff

β) i = ~0x1f84; /* Το ΣΤ1 του 1f84 ειναι e07b */
```

Ευρεση μεγιστης τιμης ενος τυπου

Για int που ειναι 2 bytes:

- Η μεγιστη θετικη τιμη που μπορει να λαβει ενας int ειναι:

```
    cout << 0x7fff;
```

- Η μεγαλυτερη ακεραια τιμη (χωρις προσημο) ειναι:

```
    cout << 0xffff;
    ή
    unsigned int i;
    i = ~0;    // ΣΤ1 για το 0 = ffff
    cout << i;
```

-- Το συμπληρωμα του 2 (ΣΤ2) ενος αριθμου k ειναι το ΣΤ1 + 1. Το ΣΤ2 ειναι ο ιδιος αριθμος αλλά αρνητικος δηλ. -k

```
#include <iostream.h>
int main()
{
    int k = 1234;

    cout << "        k = " << k << endl;
    cout << "ΣΤ1 του k = " << ~k << endl;
    cout << "ΣΤ2 του k = " << ~k + 1 << endl;

    return(0);
}
```

Αποτελεσμα

```
        k =  1234
ΣΤ1 του k = -1235
```

```
ΣΤ2 του k = -1234
```

Τελεστες Αριθμητικων Πραξεων

Οι τελεστες που χρησιμοποιουνται στις κοινες αριθμιτικες πραξεις (arithmetic operators) ειναι:

- `+` Προσθεση
- `-` Αφαιρεση
- `*` Πολ/σμο
- `/` Διαιρεση
- `%` Το ακεραιο υπολοιπο μιας διαιρεσης (modulus)
  - α) `A = x % y;`
  - β) `cout << 17 % 5;  // Θα εμφανισθει 2`

Τελεστης ++

Ο `++` ειναι τελεστης αυξησης κατα ενα (increment operator). Ο τελεστης `++` αυξανει κατα ενα την τιμη μιας μεταβλητης. Μπορει να προηγειται (προθεμα - prefix) ή να ακολουθει (απιθεμα - postfix) την μεταβλητη.

1. `i++;` Ισοδυναμο με: `i = i + 1;`
2. `++i;` Ισοδυναμο με: `i = i + 1;`

Υπαρχει διαφορα στην εκτιμηση μιας μεταβλητης με τον τελεστη `++` οταν βρισκεται μεσα σε μια παρασταση:

```
        j = i++;
        Ισοδυναμο με:
        j = i;       // Πρωτα γινεται απονομη τιμης
        i = i + 1;   // και μετα η αυξηση κατα ενα
ενώ:
        j = ++i;
        Ισοδυναμο με:
        i = i + 1;   // Πρωτα γινεται η αυξηση κατα ενα
        j = i;       // και μετα η απονομη τιμης
```

Τελεστης --

Ο `--` ειναι τελεστης μειωσης κατα ενα (decrement operator). Ο `--` μειωνει κατα ενα την τιμη μιας μεταβλητης. Μπορει να προηγειται (προθεμα - prefix) ή να ακολουθει (απιθεμα - postfix) την μεταβλητη.

1. `i--;` Ισοδυναμο με: `i = i - 1;`
2. `--i;` Ισοδυναμο με: `i = i - 1;`

Υπαρχει διαφορα στην εκτιμηση μιας μεταβλητης με τον τελεστη `--` οταν βρισκεται μεσα σε μια παρασταση:

```
    j = i--;  // Ισοδυναμο με:  j = i;
                                 i = i - 1;

ενω j = --i;  // Ισοδυναμο με:  i = i - 1;
                                 j = i;
```

Η χρηση μεταβλητων που αυξανονται ή ελλατωνονται για δευτερη φορα σε μια παρασταση (χωρις να ειναι γνωστη η προτεραιοτητα) δημιουργει ασαφη κωδικα και θα συμπεριφερεται διαφορετικα σε διαφορετικους μεταφραστες.

```
chalia = 0;
```

```
mapa = ++chalia + 2 * chalia++;
```

Και αλλοι τελεστες απονομης τιμων

```
+=   Προσθεση    i += 5;  ισοδυναμο με: i = i + 5;
-=   Αφαιρεση    i += 7;  ισοδυναμο με: i = i - 7;
*=   Πολ/σμος    i *= 4;  ισοδυναμο με: i = i * 4;
/=   Διαιρεση    i /= 2;  ισοδυναμο με: i = i / 2;
%=   Modulus     i %= 3;  ισοδυναμο με: j = j % 3;
```

πχ. `v *= n + 4;` ισοδυναμο με: `v = v * (n + 4);`

Πραξεις μεταξυ ιδιου ή διαφορετικου τυπου μεταβλητων ή σταθερων

		Παραδειγματα	
Πραξη	Αποτελεσμα	Πραξη	Αποτελεσμα
ακερ + ακερ	ακερ	4 + 7	11
πραγ + ακερ	πραγ	4.4 + 7	11.4
πραγ + πραγ	πραγ	4.5 + 7.6	12.1
ακερ * πραγ	πραγ	3 * 6.5	19.5
ακερ / ακερ	ακερ	10 / 5	2
		13 / 5	2
ακερ / πραγ	πραγ	10 / 2.5	4.0
πραγ / ακερ	πραγ	5.0 / 9	0.6

Εαν εχουμε:

```
double f;  int j = 7, i = 2;
f = j / i;
cout << f;
```

Θα εμφανισθει: `3`

Η πραξη j/i διαιρεση ειναι μεταξυ ακεραιων; το αποτελεσμα της πραξης ειναι ακεραιος και τα δεκαδικα ψηφια χανονται.

```
double m = 2.4, n = 4.78; int j;
j = m * n;    // 2.4 * 4.78 = 11.472
cout << j;
```

Θα εμφανισθει: `11`

Πολ/σμος μεταξυ δυο πραγματικων, το αποτελεσμα της πραξης ειναι πραγματικος και απονεμεται σε ακεραιο, τελικως χανονται τα δεκαδικα ψηφια.

Μετατροπες τυπων δεδομενων (casting)

Προσωρινη μετατροπη των δεδομενων γινεται με δυο τροπους:

A) Με τελεστες μετατροπης (casting operator)

```
        τυπος(σταθερα ή μεταβλητη ή παρασταση);
```

οπου ***τυπος*** ειναι ενας απο τους βασικους τυπους δεδομενων (πχ. `int, float`)

Παραδειγματα

```
α) int k = int(65.5);
   float f = float(k);
   char ch;
   ch = char(f);
```

```
cout << k << f << ch << endl;
```
Θα εμφανισθει **65 65.0 A**

β)
```
int i,j; float f;
f = j / i;
```
Εαν δεν θελουμε να χαθουνε τα δεκαδικα ψηφια η διαιρεση πρεπει να αλλαξει:
```
f = j / float(i);
```
Ο i μετατρεπεται σε float (για αυτην την πραξη); το αποτελεσμα ειναι float.

γ)
```
if (j/2 == j/2.0)    // Ελεγχος εαν ο j ειναι ζυγος
```
ισοδυναμο με
```
if (j/2 == j/float(2))
```

δ)
```
char ch;
:
if (ch == char(65))  // Ελεγχος εαν ο ch ειναι ο A
```
ισοδυναμο με
```
if (int(ch) == 65)
```

ε)
```
int i; long k;
:
k = unsigned(i);
```
Μετατροπη του i σε μη προσημασμενο long ακεραιο, μονο για αυτην την εντολη

στ)
```
int i;  unsigned int w;
 :
w = unsigned(i);
```
Ο i μετατρεπεται σε μη-προσημασμενο ακεραιο, μονο για αυτην την εντολη.

Β) Διατηρειται και στη C++ ο τροπος μετατροπης τυπων της C.

(τυπος) σταθερα ή μεταβλητη ή πρασταση;

οπου ***τυπος*** ειναι ενας απο τους βασικους τυπους δεδομενων (πχ. **int, float**)

Παραδειγματα

α)
```
int i; float f;
:
f = (float) i + 5;
```
Ο i μετατρεπεται μονο για αυτην την πραξη σε πραγματικο.

β)
```
int g; long k;
:
k = (long) g;
```

γ)
```
double f; int k = 5, m = 9;
f = k / (double) m;
```

Εμβελεια μεταβλητων

Οι μεταβλητες ειναι γνωστες (προσπελασιμες, "ζωντανες") μονο μεσα στο block που δηλωνονται.

Παραδειγματα

α)
```
if(a == 5) {
     int x;
```

```
        x = 5;
    }
    // Σε αυτο το σημειο η μεταβλητη x ειναι αγνωστη

β)  for(int i = 0; i < 20; i++)
        cout << i;
    // Σε αυτο το σημειο η μεταβλητη i ειναι αγνωστη

γ)  x = 3;       // Εξωτερικο x
    if(x > 0) {
        int x;  // Εσωτερικο x
        x = 7;
    }
    cout << "x=" << x;  // Θα εμφανισθει το εξωτερικο x=3
```

Οι μεταβλητες που οριζονται στη αρχη του πηγαιου προγραμματος (εξω απο το main) ειναι γνωστες σε ολο το πηγαιο αρχειο.

-- Παραδειγμα

```
#include <iostream.h>
int x = 3;
char ch;  // Οι μεταβλητες x και ch ειναι γνωστες σε ολο το πηγαιο προγραμμα
int main()
{
    int z; // Η μεταβλητη z ειναι γνωστη μονο μεσα στο main
    cout << "x = " << x << endl;
    return(0);
}
```

Ασκησεις

-- Να ορισετε μια ακεραια μεταβλητη, να της δωσετε αρχικη τιμη, και στην συνεχεια να την εμφανισετε με cout σε δεκαδικη (dec), οκταδικη (oct), δεξαεδικη μορφη (hex) και στην ASCII μορφη. (Δωσετε μια τιμη 65 και μετα την τιμη 12345).

-- Να δοκιμασετε εαν ο μεταφραστης (compiler) σας δεχεται την δηλωση μιας μεταβλητης ως long float και ως long double. Βρητε το μεγεθος τους κανοντας χρηση του τελεστη sizeof. (long float ειναι ισοδυναμο με double)

-- Να βρητε το μεγεθος ολων των τυπων (short, int, long, float, double, char) στον δικο σας compiler. (Βοηθημα: χρησιμοποιειστε τον τελεστη sizeof)

-- Να βρητε την μεγαλυτερη / μικροτερη τιμη για τους παρακατω τυπους: ακεραιο (int), μεγαλο ακεραιο (long), και μικρο ακεραιο (short). (Βοηθημα: Βρητε πρωτα το μηκος των τυπων σε bytes)

-- Να βρητε την μεγαλυτερη θετικη τιμη για τους παρακτω τυπους χωρις προσημο (unsigned): ακεραιο (int), μεγαλο ακεραιο (long), και μικρο ακεραιο (short).

-- Να γραψετε ενα προγραμμα στο οποιο δηλωσετε ως float δυο μεταβλητες, Arx_kefal, και epitokio. Δωσετε αρχικες τιμες στις μεταβλητες (πχ. 100000 και 20 αντιστοιχα) Να υπολογισετε τον τοκο με την σχεση
tokos = arx_kefal * epitokio / 100
Τελος να εμφανισθουν (με printf) τα μηνυματα:

```
Αρχικο Κεφαλαιο: xxxxxxxx.xx
Επιτοκιο       :       xx.xx
Τοκος          :   xxxxxx.xx
Τελικο Κεφαλαιο: xxxxxxxx.xx
```

Τελεστες Σχεσεων

Οι τελεστες σχεσεων (relational operators) ειναι:

>	Μεγαλυτερο
>=	Μεγαλυτερο ή ισο
<	Μικροτερο
<=	Μικροτερο ή ισο
==	Ισο
!=	Διαφορο (οχι ισο)

Οι τελεστες **>**, **>=**, **<**, **<=** εχουν το ιδιο βαθμο προταιρεοτητος.

Μικροτερη προτεταιτητα εχουν οι τελεστες ισοτητος **==**, **!=** (μεταξυ τους εχουν τον ιδιο βαθμο προτεραιοτητος)

Κατασκευη συνθηκης

σταθερα ή μεταβλητη ή παρασταση ΣΧΕΣΗ σταθερα ή μεταβλητη ή παρασταση

Οπου *ΣΧΕΣΗ* ειναι ενας απο τους τελεστες σχεσεων.
Η συνθηκη εκτιμαται σε αληθή (true) δηλ. != 0 (διαφορο του μηδενος) ή σε ψευδή (false) δηλ. 0 (μηδεν).
Παραδειγματα:

```
α) 5 > 3
β) i <= j
γ) k == m + 4
δ) g - 6 != k*5 + 3
```

Εντολη if-else

Η εντολη if αλλαζει την ροη εκτελεσης του προγραμματος. Η if εχει την εξης μορφη:

```
if (συνθηκη) προταση_Α;
```

Η *προταση_Α* εκτελειται μονο οταν η συνθηκη ειναι αληθης

Εαν η προταση ειναι συνθετη τοτε παιρνει την μορφη:

```
if (συνθηκη) {
     προταση_1;
     προταση_2;
     :
}
```

Το **if** μπορει να συνοδευεται απο το **else**:

```
if (συνθηκη)
     προταση_Α;
else
     προταση_Β;
```

Εαν η συνθηκη ειναι αληθης εκτελειται η *προταση_Α* αλλιως εκτελειται *προταση_Β*.

Παραδειγματα

```
α) if (15 > 4) a = 5;  // Τοτε γινεται: if (1) a = 5
```

```
β) if (5) flag = 4;     // Η flag παιρνει την τιμη 4.

γ) if(a < 0) f = g = h = 0;

δ) if(a < 0)
       i++;
   else --i;

ε) if(a < 0 && b == 0) {
       w += 4;
       r *= 3;
   }
   else if(a == 0) {
       k = (i=i+1) + 1;
   }
   else k = (i /= 2) + 1;   // δηλ. k=(i = i/2)+1

στ) if (i = (15 > 9)) j = 3;     // Το i γινεται 1
    cout << "i = " << i << "   j = " << j << endl;
    // Εμφανιζεται το i = 1   j = 3
```

Το **else** ειναι προαιρετικο και σχετιζεται μετο πλησιεστερο if.
Παραδειγμα

```
if (n > 0)
    if (a > b)
        z = a;
    else
        z = b;
```

Το else συνδεεται με το εσωτερικο if. Εαν δεν ειναι αυτο που θελετε τοτε επιβαλεται η χρηση των αγκιστρων (δημιουργια block). Ετσι το προηγουμενο παραδειγμα γινεται:

```
if (n > 0) {
    if (a > b)
        z = a;
}
else
    z = b;
```

Ακομη ενα παραδειγμα (παραδειγμα κακης γραφης κωδικα προγ/σμου)

```
if (a != b)
if (g == 5)
i++;
else
j++;
else
cout << "Adam - Eva";
```

Το ιδιο παραδειγμα με καλυτερη γραφη:

```
if (a != b)          // 1ο if
    if (g == 5)            // 2ο if
        i++;
    else                   // 2ο else
        j++;
```

```
    else                    // 1ο else
        cout << "Adam - Eva";
```

Λογικοι Τελεστες

Οι λογικοι τελεστες (boolean or logical operators) ειναι

`&&`	λογικο	ΚΑΙ	(and)
`||`	λογικο	Ή	(or)
`!`	λογικο	ΟΧΙ	(not)

Παραστασεις (expressions) που συνδεονται με `&&` ή `||` εκτιμωνται απο αριστερα προς τα δεξια.

`--` Παραδειγμα

```
#include <iostream.h>
int main() {
    int a,b;
    a = 0; b = 5;

    // Προσοχη! Το b = 2 ειναι απονομη τιμης και ΟΧΙ συγκριση
    //
    if (a == 1 && (b = 2))// Το b θα κρατησει την τιμη 5; η
                          // εκτελεση του if τελειωνει
                          // επειδη a==1 ειναι ψευδης (false)
        cout << "b = " << b;
    else
        cout << "b = " << b;

    return(0);
}
// p15.cpp //
```

Αποτελεσμα

```
B = 5
```

Ασκηση. Στο προηγουμενο προγραμμα τι θα συμβει αν το **a** λαβει αρχικη τιμη 1 αντι για 0.

Ο τελεστης `!` αντιστρεφει την συνθηκη (εαν ειναι αληθης την μετατρεπει σε ψευδη, και εαν ειναι ψευδης σε αληθη)

```
i = 52;
if (!(i > 23))
    εντολη_Α;
else
    εντολη_Β;
```

`//` Δεν θα εκτελεσθει η *εντολη_Α* αλλα η *εντολη_Β*

`--` Παραδειγμα

```
#include <iostream.h>
int main()
{
   int a, b = 2;
   if (!(b == 2))
       a = 3;
```

```
   else
       a = 1;
   cout << "a = " << a << endl;
   return(0);
}
```
Αποτελεσμα
```
a = 1
```

Οι τελεστες σχεσεων εχουν μικροτερη προτεραιοτητα απο τους τελεστες αριθμητικων πραξεων.

```
if (i > j - 1)   // Πρωτα γινεται η αφαιρεση και μετα η συγκριση.
```

Εντολη `switch`

Η εντολη switch αλλαζει την ροη εκτελεσης του προγραμματος.

```
switch (παρασταση) {
    case τιμη_A:
        εντολες_A
        break;
    case τιμη_B:
        εντολες_B
        break;
    case τιμη_C:
        εντολες_C
    default:
        εντολες_D
}
```

Η *παρασταση* εξεταζεται (διαδοχικα) με τις τιμες στα case δηλ. εαν η παρασταση ειναι ιση με την *τιμη_A*, με την *τιμη_B* κοκ. Εαν η παρασταση ειναι ιση με μια τιμη τοτε εκτελουνται οι ολες εντολες που υπαρχουν κατω απο το case εκτος εαν εκτελεσθει η break οποτε τελειωνει η εκτελεση του switch.

- Το **`default`** ειναι προαιρετικο και εκτελειται οταν κανενα case δεν εκτελειται.
- Η εντολη **`break`** ειναι προαιρετικη και διακοπτει την εκτελεση της **`switch`**.
- Η *παρασταση* και η *τιμες* πρεπει να ειναι ακεραιοι τυπου char, short, int, long.
- Η *εντολες_A* εκτελουνται οταν η *παρασταση* εχει τιμη ιση με την *τιμη_A*.
- Η *εντολες_B* εκτελουνται οταν η *παρασταση* εχει τιμη ιση με την *τιμη_B*.
- Η *εντολες_C* και *εντολες_D* εκτελουνται οταν η *παρασταση* εχει τιμη ιση με την *τιμη_C* (εαν δεν θελουμε να εκτελουνται οι *εντολες_D* εισαγουμε την εντολη break)
- Η *εντολες_D* εκτελουνται οταν η παρασταση δεν ιση ουτε με την *τιμη_A*, ουτε με την *τιμη_B* ουτε με την *τιμη_C*.

Παραδειγματα

```
1) int i;
   i = 2;
   switch(i) {
       case 2:
          cout << i*2;
          break;
       case 9:
          cout << i*9;
```

```
}
```
Θα εμφανισθει **4**

```
2) int i;
   :
   switch(i+5) {
      :
   }

3) long k;
   :
   switch(k*5+2) {
      :
   }

4) int i;
   :
   switch (i) {
       case 1: cout << "Male  "; break;
       case 2: cout << "Female"; break;
       default:
           cout << "Error";
   }
```
- Εαν το i ειναι 1 τοτε θα εμφανισθει το μηνυμα: **Male**
- Εαν το i ειναι 2 τοτε θα εμφανισθει το μηνυμα: **Female**
- Για τιμες του i που δεν ειναι 1 ή 2 θα εντελεσθει η default και θα εμφανισθει το μηνυμα: **Error**

```
5) char ch; int k = 0;
   :
   ch = 'B';
   switch (ch) {
      case 'A':  // Εαν ch='A' τοτε θα εκτελεσθει η k=4
         k = 1;
         break;
      case 'B':  // Εαν 'B' ή 'C' θα εκτελεσθει η k=23
      case 'C':
         k = 23;
         break;
      default:     // Η k=678 θα εκτελεσθει για ολες τις
         k = 678;  // υπολοιπες τιμες του ch.
   }
```

Θηλειες (loops)

Α) Θηλεια με **for**
Β) Θηλεια με **while**
Γ) Θηλεια με **do**

Α) Η θηλεια **for** εχει την μορφη

```
for(μ_θ = αρχικη_τιμη; συνθηκη; μ_θ = μ_θ + βημα)
    προταση;
```

(οπου *μ_θ = μεταβλητη_θηλειας*)

Εαν η *προταση* ειναι συνθεση εντολων τοτε εχει την μορφη

```
for(μ_θ = αρχικη_τιμη; συνθηκη; μ_θ = μ_θ + βημα) {
   εντολη_1;
   εντολη_2;
     :
   εντολη_N;
}
```

Η **for** εκτελειται συμφωνα με τα ακολουθα βηματα:
1. Υπολογισμος αρχικης τιμης της μεταβλητης θηλειας. Ο υπολογισμος γινεται μια φορα.
2. Εαν η συνθηκη ειναι ψευδης τελειωνει ολη η προταση for.
3. Εκτελεση εντολων for
4. Η μεταβλητη θηλειας αυξανεται κατα το βημα και πηγαινει στο βημα 2.

α)
```
for(int i = 1; i <= 4; i++)
    cout << i << " ";
```
Αποτελεσμα
```
1 2 3 4
```

β)
```
for(i = 1; i < 5; i = i + 1)
    cout << char('A'+i);
```
Αποτελεσμα
```
BCDE
```

γ)
```
for(i = -15; i <= 15 ; i++)
    cout << i << " ";
```
Θα εμφανισθει η σειρα:
```
-15 -14 -13 ... 2 -1 0 1 2 ... 13 14 15
```

```
#include <iostream.h>
int main()
{
    cout << "AAA";
    for( k = 1; k < 5 ; k++)
        cout << '!';
    return(0);
}
```

Αποτελεσμα

```
AAA!!!!
```

Β) Η θηλεια **while** εχει την μορφη:

```
while ( συνθηκη ) προταση;
```

Εαν η *προταση* ειναι συνθετη τοτε παιρνει την μορφη:

```
while ( συνθηκη ) {
     εντολη_1;
     εντολη_2;
       :
     εντολη_N;
}
```

Η **while** εκτελειται συμφωνα με τη ακολουθη διαδικασια:
1. Εκτιμαται η συνθηκη.
2. Εαν η συνθηκη ειναι ψευδης (false) εκτελειται η πρωτη προταση μετα το while.
3. Εαν η συνθηκη ειναι αληθης (true) εκτελειται η προταση μεσα στο while
4. Πηγαινει στο βημα 1.

α)

```
i = 5;
while (i > 0) cout << i-- << " ";
```

Αποτελεσμα

```
5 4 2 1
```

β)

```
i = 1;
while(i < 10) {
    i += 3;
    cout << i << " ";
}
```

Θα εμφανισθει:

```
1 4 7
```

```
#include <iostream.h>
int main()
{
    int k = 1;
    while( k <= 4 )
       cout << 2*k << " " << endl;
    return(0);
}
```

Αποτελεσμα

```
2 4 6 8
```

Γ) Η θηλεια **do** εχει την μορφη

```
do
   προταση;
while ( συνθηκη );
```

Εαν η *προταση* ειναι συνθετη τοτε παιρνει την μορφη:

```
do {
     εντολη_1;
     εντολη_2;
       :
     εντολη_N;
} while ( συνθηκη );
```

Η διαδικασια εκτελεσης της θηλειας **do** ειναι:
1. Εκτελουνται ολες οι εντολες μεσα στην θηλεια.
2. Γινεται εκτιμηση της συνθηκης.
3. Εαν η συνθηκη ειναι ψευδης τελειωνει η εκτελεση της θηλειας.
4. Εαν η συνθηκη ειναι αληθης τοτε πηγαινει στο βημα 1.

```
i = 0;
do
   cout << i << " ";
while (i++ < 5); // Πρωτα γινεται η συγκριση και μετα η αυξηση κατα 1
```
Εμφανιζει: **1 2 3 4 5**

```
#include <iostream.h>
int main()
{
    int i = 1;
    do {
        cout << i << " ";
        i += 3;
    } while (i <= 10);
    return(0)
}
```
Θα εμφανισθει:
1 4 7 9

Η διαφορά μεταξυ της **while** και της **do** θηλειας ειναι οτι η θηλεια do εκτελειται τουλαχιστον μια φορα ανεξαρτητως εαν η συνθηκη ειναι αληθης ή ψευδης.

Οι παρακατω θηλειες ειναι ατερμονες (χωρις τελος) διοτι η συνθηκη ειναι παντα αληθης:
```
α) for(;;) { εντολες };
b) while (1) { εντολες };
c) do { εντολες } while (1);
```

break και continue

Εντολες της C/C++ που συνδεονται με τις θηλιες (for, while, do) ειναι η **break** και η **continue**.

- Η εντολη **break** τερματιζει την εκτελεση της θηλειας.
- Η **continue** "πηδά" στην αρχη της θηλειας.

Παραδειγμα χρησης **continue**
```
while (1) {          // Οταν το a == 0 τοτε δεν θα εκτελεσθει
    προταση_Α;       // η εντολη_Β αλλα θα "πηδηξει" στο while.
    if (a == 0)
        continue;
    προταση_Β;
}
```

Παραδειγμα χρησης **break**

```
while (1) {       // Οταν το  a == 0  τοτε δεν θα εκτελεσθει
    προταση_A;    // η προταση_B αλλα θα "πηδηξει" στην πρωτη

    if (a == 0)   // προταση_C μετα το block του  while.
        break;
    προταση_B;
}
προταση_C;
```

Τελεστης ?:

Συνταξη:

```
( συνθηκη  ?  Exp1  :  Exp2 )
```

οπου ***Exp*** μπορει να ειναι σταθερα ή μεταβλητη ή παρασταση

Εαν η *συνθηκη* ειναι αληθης τοτε ολη η εντολη εκτιμαται ιση με την *Exp1* διαφορετικα η εντολη εκτιμαται ιση με την *Exp2*.

Παραδειγματα

```
α) a = 7;
   :
   x = (a == 7 ? 12 : 3);    // Το x παιρνει την τιμη 12

β) y = 6;
   :
   cout << (y > 0 ? "Alpha" : "Beta");
```

Θα εμφανισθει η λεξη **Alpha**

```
γ) r = (x % y ? sqrt(y) : abs(x));
```

Ασκησεις

-- Να γραψετε ενα προγραμμα το οποιο μετατρεπει σε θερμοκρασια Celsius μια θερμοκρασια Fahrenheit. (Βοηθημα: cel = (5/9)*(fahr-32))
Προσοχη: η διαιρεση 5/9 δινει ακεραιο αριθμο και το αποτελεσμα θα ειναι παντα μηδεν, εκτος και αν χρησιμοποιεισετε 5/9.0 ή 5.0/9.0 ή 5/float(9)

-- Να γραψετε ενα προγραμμα το οποιο θα εμφανιζει τον παρακατω πινακα με τις θερμοκρασιες Fahrenheit με τις αντιστοιχες θερμοκρασιες Celsius.

```
Fahr     Cel
  0     -17.8
 20      -6.7
 40       4.4
...       ...
260     126.7
280     137.8
300     148.9
```

Να χρησιμοποιεισετε θηλεια με **while.**

Λυση 1:

```
#include <iostream.h>
#define  MIKROTERI  0
#define  MEGALYTERI 300
#define  BHMA       20
int main()
{
    int fahr;
    cout << "Fahr    Celcius" << endl;
    for(fahr=MIKROTERI; fahr<=MEGALYTERI; fahr=fahr+BHMA)
        cout << fahr << "  " <<  (5.0/9.0)*(fahr - 32.0)
             << endl;
    return(0);
}
// p10.cpp //
```

Λυση 2:

```
#include <iostream.h>
int main()
{
    int fahr;
    float cel;

    cout << "Fahr    Celcius\n";
    fahr = 0;
    while(fahr <= 300) {
        cel = (5.0/9.0)*(fahr - 32.0);
        cout << fahr << "   " << cel << endl;
        fahr = fahr + 20;
    }
    return(0);
}
// p11.cpp //
```

-- Να γραφει ενα προγραμμα το οποιο διαβαζει απο το τερματικο σε μια γραμμη (σε ενα cin) εναν int, εναν float, εναν char, και ενα string.

```
#include <stdio.h>
int main()
{
    int i; float f; char ch; char s[30];

    cin >> i >> f >> ch >> s;

    return(0)
}
// p11a.cpp //
```

-- Στην προηγουμενη ασκηση να χρησιμοποιειστε αντι της cin την scanf.

-- Να γραφει ενα προγραμμα το οποιο διαβαζει απο το τερματικο ενα αριθμο int. Εαν ο αριθμος που διδεται ειναι εκτος περιοχης -120 εως +120 τοτε να εμφανιζεται αντιστοιχο μηνυμα.
Παραλλαγη: Εαν ο αριθμος που εχει δωθει ειναι αρνητικος τοτε να μετατραπεται σε θετικο και εμφανιζεται αντιστοιχο μηνυμα.

-- Να γραφει ενα προγραμμα το οποιο διαβαζει απο το τερματικο ενα αριθμο int. Εαν ο αριθμος που διδεται ειναι εκτος περιοχης -120 εως +120 και μονός τοτε να εμφανιζεται αντιστοιχο μηνυμα ή αριθμος εκτος περιοχης ή αριθμος μονός.

-- Να γραφει ενα προγραμμα το οποιο διαβαζει απο το πληκτρολογιο ακεραιους αριθμους που αντιπροσωπευουν ημερησιες μεσες θερμοκρασιες. Το διαβασμα τελειωνει με τον αριθμο -999. Επιτεπομενες θερμοκρασιες -50 εως και +60.
Με το τελος των αριθμων να εμφανιζονται:
1. Το συνολο των αριθμων (ημερων) που δωθηκαν.
2. Η μεση θερμοκρασια.
3. Το συνολο των ζεστων ημερων (θερμοκρασιες απο 18 - και 28) και η μεση θερμοκρασια.
3. Το συνολο των κρυων ημερων (θερμοκρασιες απο 1 - και 17) και η μεση θερμοκρασια.
4. Το συνολο των παγωμενων ημερων (θερμοκρασιες απο -1 και κατω) και η μεση θερμοκρασια.
5. Το συνολο των ιδανικων ημερων (θερμοκρασια 22).
Παραλλαγη: Οταν ο χρηστης δωσει το -999 να ερωτηθει εαν θελει να τελειωσει το διαβασμα των θερμοκρασιων με (Y/N). Μονο οταν ο χτηστης δωσει το Y (ναι) να τελειωνει το διαβασμα (ολες οι αλλες επιλογες θα θεωρουνται N = οχι).

-- Να γραφει ενα προγραμμα το ποιο ελεγχει εναν χαρακτηρα. Ο χαρακτηρας διαβαζεται απο το τερματικο και στην συνεχεια εμφανιζει ενα απο τα μηνυματα:
a) Ο χαρακτηρας ειναι αλφαβητικος αν ανοικει στις περιοχες A..Z, a..z.
b) Ο χαρακτηρας ΔΕΝ ειναι αλφαβητικος.
Επισης ενα απο τα μηνυματα:
a) Ο χαρακτηρας ειναι γραμμα κεφαλαιο εαν ειναι στην περιοχη A εως και Z
b) Ο χαρακτηρας ειναι γραμμα μικρο εαν ειναι στην περιοχη a εως και z
c) Ο χαρατηρας ειναι αριθμητικος 0 εως 9.
d) Ο χαρακτηρας ειναι κοντρολ εαν ειναι μικροτερος του ' ' (κενο=space)

e) Ο χαρακτηρας ειναι ανηκει στους ειδικους χαρακτηρες *&%$ κλπ.
Το προγραμμα να σχεδιασθει να ελεγχει πολλους χαρακτηρες (δημιουργια θηλειας). Τελος προγραμματος εαν δωθει ο χαρακτηρας Q ή q.

```
#include <iostream.h>
int main()
{
    char ch;

    while (1) {
        cin >> ch;

        if(ch == 'Q' || ch == 'q')
            return(0);
            /*
        Τερματισμος προγραμματος.  Η main ειναι μια συναρτηση
        που καλειται απο το prompt του λειτουργικου. Με το
        return επιστρεφει τον ελεγχο στην καλουσα συναρτηση.
        Στην προκειμενη περιπτωση στο Λ.Σ.
            */

        if((ch >= 'A' && ch <= 'Z') || (ch >= 'a' &&
                   ch <= 'z'))
             cout << "Ο χαρακτηρας: " << ch
                  << " ειναι αλφαβητικος\n";
        else cout << "Ο χαρακτηρας: " << ch <<
                     " ΔΕΝ ειναι αλφαβητικος" << endl;

        if(ch >= 'A' && ch <= 'Z')
             cout << "Ο χαρακτηρας: " << ch <<
                     " ανοικει στην περιοχη A..Z" << endl;
        else if(ch >= 'a' && ch <= 'z')
             cout << "Ο χαρακτηρας: " << ch <<
                     " ανοικει στην περιοχη a..z\n";
        else if(ch >= '0' && ch <= '9')
             cout << "Ο χαρακτηρας: " << ch <<
                     " ειναι αριθμητικος" << endl;
        else if(ch < ' ')
             // Εμφανιση του χαρακτηρα σε hex μορφη
             cout << "Ο χαρακτηρας " << hex << ch <<
                     " ειναι control χαρακτηρας\n";
        else cout << "Ο χαρακτηρας: " << ch <<
                     " ειναι ειδικος\n";

        cout << "Πιεστε <Enter> ή <Return> ...";
    }
}
// p12.cpp //
```

-- Να γραφει ενα προγραμμα το οποιο διαβαζει εναν χαρακτηρα και στην συνεχεια ελεγχεται για τα ακολουθα:
a) εαν ο χαρακτηρας ειναι ο A να μετατραπει σε M.
b) εαν ο χαρακτηρας ειναι ο B να μετατραπει σε b.
c) εαν ο χαρακτηρας ειναι ο C να μετατραπει σε E (αυτο το E δεν επιτρεπεται να αλλαξη και παλι)

d) εαν ο χαρακτηρας ειναι ο D να μετατραπει σε E (αυτο το E επιτρεπεται να αλλαξη και παλι)
e) εαν ο χαρακτηρας ειναι ο E ή T ή Y ή U να μετατραπει σε '@'.
f) εαν ο χαρακτηρας ειναι ο 4 τοτε να μετατραπει σε 7.
h) διαφορετικα ο χαρακτηρας θα παραμεινει ως εχει.
Ο μεταλλαγμενος ή οχι χαρακτηρας θα εμφανιζεται στην οθονη. Εαν ο χαρακτηρας ειναι control τοτε να εμφανιζεται η δεκαεξαδικη του μορφη.
Το προγραμμα να σχεδιασθει να ελεγχει πολλους χαρακτηρες (δημιουργια θηλειας). Τελος προγραμματος εαν δωθει ο χαρακτηρας Q ή q.

```
#include <iostream.h>

#define TRUE  1
#define FALSE 0

int main()
{
    char ch;
    int ctrl;

    ctrl = FALSE;
    while (1) {
        cin >> ch;
        if(ch == 'Q' || ch == 'q')
            return(0);
        if(ch < 32) ctrl = TRUE;    // control χαρακτηρας

        switch (ch) {
            case 'A':
                ch = 'A' + 12;   // ιδιο με ch = 'M'
                break;
            case 'B':
                ch = 'b';
                break;
            case 'C':
                ch = ch + 2;     /* ch = 'E' = 'C' + 2 */
                break;
            case 'D':
                ch = 'E';
            case 'E':
            case 'T':
            case 'Y':
            case 'U':
                ch = 64;  /* '@' = 64 */
                break;
            case '4':
                ch = '7';
                break;
        }

        if(!ctrl)
             cout << "Τελικως ο χαρακτηρας ειναι " << ch
                  << endl;
        else cout << "Ο χαρακτ. σε δεκαεξαδικη μορφη ειναι "
                  << hex << int(ch) << endl;
```

```
        cout << "Τελος με Q ή q ..." << endl;
    }
    return(0);
}
// p13.cpp //
```

Παραλλαγη: Αν ο χαρακτηρας αλλαζει να εμφανιζεται το μηνυμα:
`"Ο αρχικος χαρακτηρας Χ αλλαξε."`

-- Να γραφει ενα προγραμμα το οποιο διαβαζει εναν χαρακτηρα και μετατρεπει σε αριθμους τους αριθμητικους χαρακτηρες (Ολοι οι αλλοι χαρακτηρες απορριπτομται):
a) εαν ειναι ο 1 να μετατρεπεται σε 2
b) εαν ειναι ο 3 ή 4 να πολλαπλασιαζεται με το 100.1
c) εαν ειναι ο 5 ή 6 ή 9 να προστηθεται στο 120
d) για ολους τους υπολοιπους αριθμητικους χαρακτηρες να προστιθεται ο αριθμος 7.
Τελος να εμφανιζεται το αποτελεσμα της πραξης.
Το προγραμμα να σχεδιασθει να ελεγχει πολλους χαρακτηρες (δημιουργια θηλειας).
Τελος προγραμματος εαν δωθει ο χαρακτηρας Q ή q.

```
#include <iostream.h>
#define TRUE  1
#define FALSE 0

#define digit(x)  (x >= '0' && x <= '9' ? TRUE : FALSE)


int main()
{
    char ch,dummy;
    int i;
    float res;

    while (1) {         // Συνθηκη παντα αληθης
        cout << "Δωσε εναν αριθμητικο χαρακτηρα "
                "(Q ή q = Τελος): ";
        cin >> ch;

        if(ch == 'Q' || ch == 'q')
            return(0);

        if(!digit(ch)) {  // Το πληκτρο δεν ειναι αριθμητικο
            cout << "Λαθος πληκτρο." << endl;
            continue;
        }

        i = int(ch) - 48;
        /*
           Μετατροπη αριθμητικου χαρακτηρα (ASCII) σε
           ακεραιο.  πχ. το 2 στον πινακα ASCII ειναι 50. για
           να παρουμε τον αριθμο 2 = 50 - 48
        */
```

```
        switch (i) {
            case 1:
                res = 2 ;
                break;
            case 3:
            case 4:
                res = i * 100.1;
                break;
            case 5:
            case 7:
                res = i + 120;
                break;
            default:
                res = i + 7;
        }
        cout << "Αποτελεσμα πραξης: " << res << endl;
        cout << "Πιεστε <Enter> ή <Return> ..." << endl;
    }
    return(0);
}
// p14.cpp //
```

Παραλλαγη: να ορισθει με define μια συναρτηση (με τον ιδιο προπο οπως η digit) που θα μετατρεπει τους αριθμητικους χαρακτηρες σε αριθμους. Εαν ο χαρακτηρας δεν ειναι αριθμητικος τοτε να δινει -1.

-- Υπολογισετε την τετραγωνικη ριζα του 2 με ακριβεια 0.5e-6 με την βοηθεια της μεθοδου Newton-Raphsons:

$X_0 = 1$

$X_n = 1/2*(X_{n-1} + 2/X_{n-1})$, οπου $n=1,2,3,...$

Λυση:

```
#include <iostream.h>

#define AKRIBEIA 0.5e-6
#define abs(x) (x < 0 ? -x : x)

int main()
{
    double x0,x1,diaf;

    x0 = 1.0;
    do {
        x1 = 0.5*(x0 + 2.0/x0);
        diaf = abs(x1 - x0);
        x0 = x1;
    } while (diaf > AKRIBEIA);
    cout << "Τετραγωνικη ριζα του 2 = " << x1 << endl;
    return(0);
}
// p16.cpp //
```

-- Να εμφανισετε τις ακολουθες σειρες. Χρησιμοποιεισετε θηλεια for.

```
α) 1  2  3  4 ... 24  25
β) 1  -2  3  -4  5 ... -24  25
γ) 1  -1  2  -2  3  -3 ... 25 -25
```

Λυση της α)

```
#include <iostream.h>
int main()
{
    int i;
    for(i = 1; i <= 25; i++)
        cout << i << " ";
    return(0);
}
```

-- Να γραφει ενα προγραμμα το οποιο σχεδιαζει στην οθονη ενα ορθογωνιο τριγωνο με αστερακια ('*'). Χρησιμοποιεισετε θηλειες for.

```
α)       *         β)  *****
        **                ****
       ***                 ***
      ****                  **
     *****                   *
```

-- Να γραφει ενα προγραμμα το οποιο σχεδιαζει στην οθονη ενα ισοσκελες τριγωνο με αστερακια ('*'). Χρησιμοποιεισετε θηλειες for.

```
          *
         ***
        *****
       *******
      *********
```

-- Να γραφει ενα προγραμμα το οποιο να μετρα το πληθος των χαρακτηρων που διαβαστηκαν με την getchar.

```
#include <stdio.h>
#define EOL '\n'   /* End-Of-Line */

int main()
{
   char ch; int cnt = 0;

   while ((ch = getchar()) != EOL)
         cnt = cnt + 1;
   printf("Πληθος χαρακτηρων που διαβαστηκαν=%d\n",cnt);
}
// ask13.c //
```

-- Να γραφει ενα προγραμμα το ποιο διαβαζει απο το τερματικο χαρακτηρες. Το διαβασμα τελειωνει με <Enter> ή <Return>. Να εμφανισθουν:
α) το πληθος των αλφαριθμητικων χαρακτηρων που διαβαστηκαν
β) το πληθος των χαρ. ελεγχου (control characters)
γ) το πληθος των ειδικων χαρακτηρων (. ; / ' [ ] \ κλπ).
(Να χρησιμοποιειθει η getchar)

```
#include <iostream.h>
#include <stdio.h>
#define EOL '\n'   /* End-Of-Line */
```

```
int main()
{
    char ch;
    int ma,mc,ms;
    ma = mc = ms = 0;  // Μηδενισμος μετρητων
    while ((ch = getchar()) != EOL) {
         if ((ch >= 'A' && ch <= 'Z') ||
             (ch >= 'a' && ch <= 'z') ||
             (ch >= '0' && ch <= '9'))
              ma++;  // μετρητης αλφαριθμητικων χαρ.
         else if(ch < ' ') mc++; // μετρητης χαρ. ελεγχου
         else ms++;  // μετρητης ειδικων χαρ.
    }
    cout << "Πληθος αλφαριθμητικων χαρακτηρων=" << ma
         << endl;
    cout << "  ..   control           ...    =" << mc
         << endl;
    cout << "  ..   ειδικων           ...    =" << ms
         << endl;
    return(0);
}
// p17.cpp //
```

-- Να εμφανισετε τους πρωτους 20 αριθμους μιας σειρας (Fibonacci) οπου ο καθε αριθμος ειναι το αθροισμα των δυο προηγουμενων:
1 1 2 3 5 8 13 κοκ.

```
#include <iostream.h>
main()
{
    int i,n1,n2,n3;
    n1 = 1;
    n2 = 1;
    cout << n1 << "  " << n2;  // Εμφανιση των δυο πρωτων
                               // αριθμων
    for(i = 1; i <= 20; i++) {
        n3 = n1 + n2;         // Ο επομενος ισος με το
                              // αθροισμα των δυο προηγουμενων
        cout << n3 << " ";
        n1 = n2;                // Μετατοπιση των αριθμων
        n2 = n3;
    }
    return(0);
} // p18.cpp
```

-- Να υπολογισθει το αρθροισμα των αριθμων απο το 1 εως το 100.

```
#include <iostream.h>
int main()
{
    int i,sum = 0;
    for(i = 1; i <= 100; i++)
         sum = sum + i;
    cout << "Αθροισμα: " << sum << endl;
    return(0);
} // p18a.cpp
```

-- Να εμφανισετε εναν πινακα με το n! δηλ. 1*2*3...(n-1)*n Το n θα διδεται απο το τερματικο.

```
#include <iostream.h>
int main()
{
    int i,n,par;

    cin >> n;
    cout << "  n       n! \n";
    cout << "-------------\n";

    par = 1;
    for(i = 1; i <= n; i++) {
        par = par * i;
        cout << i << "    " << par << endl;
    }
    return(0);
}
```

Πινακες

Για την επεξεργασια μεγαλου πληθους δεδομενων ιδιου τυπου συχνα χρησιμοποιουνται πινακες (arrays, vectors, matrix (matrices)). Ενας πινακας ειναι ενα παραδειγμα δομης-δεδομενων δηλ. η συγκεντρωση ενος συνολου δεδομενων και η οργανωση τους με καποιο σχεδιο.

Ενας πινακας αποτελειται απο ενα πληθος στοιχειων (δεδομενων), οπου καθε στοιχειο ειναι του ιδιου τυπου. Για να εντοπισθει ενα στοιχειο πρεπει να δωθει ενας δεικτης που διδει την θεση του στοιχειου στον πινακα.

Πινακες μιας διαστασης

Οι πινακες μιας διαστασης (single-dimension arrays) οριζονται ως εξης:

```
τυπος_πινακα  ονομα_πινακα[αριθμος_κελλιων];
```

```
α) int a[10];      Ο πινακας a μπορει να "κρατησει" 10 αριθμους ακεραιους.
β) long b[99];     Ο πινακας b μπορει να "κρατησει" 99 μεγαλους ακεραιους
γ) char c[25];     Ο πινακας c μπορει να "κρατησει" 25 χαρακτηρες
                   δηλ. 24_χαρ + 1_null_χαρακτηρα.
```

- Το πρωτο στοιχειο του πινακα βρισκεται στο κελί με δεικτη 0.
- Το ονομα του πινακα ειναι η διευθυνση του πρωτου στοιχειου του πινακα.

```
int a[5];
:
cout << a << endl;     // Θα εμφανισθει: 0x5eef0
printf("%x\n",&a[0]); // Θα εμφανισθει:   5eef0
```

Ενα παραδειγμα πινακα μιας διαστασης οπου τα στοιχεια του πινακα ειναι οι 10 πρωτοι αρριθμοι:

```
int ar[10];
```

στοιχεια:	2	3	5	7	11	13	17	19	23	29
δεικτες:	0	1	2	3	4	5	6	7	8	9

```
ar[4] = 11;
```
Απονομη τιμης στο κελλι με δεικτη 4

```
cout << ar[8];
```
Ανακτηση τιμης του κελλιου με δεικτη 8

Πινακες δυο διαστασεων

Δηλωση πινακων δυο διαστασεων (two-dimensional arrays):

```
τυπος_πινακα ονομα_πινακα[αριθμος_γραμμων][αριθμος_στηλων];
```

```
int a[4][5];
```
Πινακας ακεραιων με 4 γραμμες και 5 στηλες.

```
float b[6][50];
```
Πινακας float με 6 γραμμες και 50 στηλες.

```
char s[24][80];
```
Πινακας char με 24 γραμμες και 80 στηλες.

- Το πρωτο στοιχειο του πινακα βρισκεται στο κελί με δεικτες [0][0]

- Το ονομα του πινακα ειναι η διευθυνση του πρωτου στοιχειου του πινακα.

```
int b[5][8];
:
cout << b << endl;          // Θα εμφανισθει: 0x5ee4c
printf("%x\n",&b[0][0]); // Θα εμφανισθει:    5ee4c
```

Ο παρακατω πινακας ακεραιων ειναι δυο διαστασεων εχει 7 γραμμες και 5 στηλες. Καθε κελι εχει δυο δεικτες: δεικτη γραμμης και δεικτη στηλης.
πχ. το στοιχειο [2][4] εχει τιμη 45.

		Σ Τ Η Λ Ε Σ				
		0	1	2	3	4
Γ	0	3	70	32	7	21
Ρ	1	23	6	2	55	12
Α	2	19	15	9	34	45
Μ	3	93	10	29	88	62
Μ	4	21	44	69	54	96
Ε	5	77	63	3	9	15
Σ	6	12	91	89	36	45

Τα στοιχεια του πινακα (οπως και οι μεταβλητες) δεν εχουν αρχικη τιμη οταν δηλωνονται (βεβαιως εχουν καποια τιμη η οποια ομως ειναι τυχαια)

Αρχικες τιμες σε πινακες

Κατα την δηλωση των πινάκων μπορουμε να δωσουμε αρχικες τιμες.
Παραδειγματα

```
α) float values[5] = { 23.4, 3.0, 5.9, 12.34, 7.69 };
β) char let[6] = { 'a', 'b', 'c', 'd', 'e', 'f' };
γ) char s[5] = "ΑΔΑΜ"; // Ο μεταφραστης προσθετει
                        στο s[4] το '\0'
δ) int sqr[2][3] = { 5, 8, 7, 0, 9, 2 };
   ή
   int sqr[2][3] = {
                      {5, 8, 7},
                      {0, 9, 2}
                   };
```

Πολλες φορες οταν δεν ειναι γνωστο το μεγεθος του πινακα τοτε ο compiler βρισκει το μεγεθος του πινακα.

```
α) int prim[] = { 1, 2, 3, 5, 7, 11 };
```

Ιδιο με: `int prim[6]`

```
β) char s[] = "Hello";
```

Στην προκειμενη περιπτωση θα καταλαβει 5+1 (5=χαρ της λεξης Hello + 1 για τον τερματικο χαρακτηρα '\0' null.

Παραδειγμα μηδενισμου ενος πινακα μιας διαστασης. Η τιμη που διδεται σε καθε στοιχειο ειναι το 0:

```
int i, a[25];
for(i = 0; i < 25; i++)
    a[i] = 0;
```

Παραδειγμα μηδενισμου ενος πινακα δυο διαστασεων.

```
int i, j, a[10,20];
for(i = 0; i < 10; i++)
    for(j = 0; j < 20; j++)
        a[i][j] = 0;
```

Ασκησεις

-- Να γραφει ενα προγραμμα το ποιο διαβαζει 15 αριθμους απο το τερματικο, τους αποθηκευει σε εναν πινακα και στη συνεχεια εμφανιζει τους αριθμους με αντιστροφη σειρα απο αυτην που διαβαστηκαν (αν δωθουν οι αριθμοι 6,33,54,12 να εμφανισθουν με την σειρα 12,54,33,6).

```
#include <iostream.h>
#define PST 15   // Πληθος στοιχειων του πινακα
int main()
{
    int i,ar[PST];

    // διαβασμα των αριθμων
    for(i = 0; i < PST; i++)
        cin >> ar[i];

    // εμφανιση με αντιστροφη σειρα
    for(i = PST-1; i >= 0; i--)
        cout << ar[i] << " ";
    return(0);
}
// p20.cpp //
```

-- Να γραφει ενα προγραμμα το οποιο διαβαζει 10 αριθμους και εμφανιζει οσους ειναι μικροτεροι απο την μεση τιμη των 10 αριθμων.

```
#include <iostream.h>
int main()
{
    int i,ar[10];
    double sum,mt;

    for(i=0; i<10; i++)
        ar[i] = 0;      // μηδενισμος του πινακα

    // διαβασμα και αθροιση των αριθμων
    sum = 0;
    for(i = 0; i < 10; i++) {
         cin >> ar[i];
         sum = sum + ar[i];
    }
    mt = sum/10;      // μεση τιμη
    cout << "Μεση τιμη=" << mt << endl;

    i = 0;


    while(i < 10) {
        // Εμφανιση εαν ar[i] < mt
        if (ar[i] < mt)
            cout << ar[i] << " ";
        i++;
    }
    return(0);
} // p21.cpp //
```

-- Να γραφει ενα προγγραμμα το ποιο διαβαζει απο το τερματικο αριθμους ακεραιους απο 0 εως και 9. Το διαβασμα των αριθμων τελειωνει με εναν αρνητικο αριθμο. Να εμφανισθει η συχνοτητα των αριθμων που διαβαστηκαν με την μορφη ιστογραμματος. πχ. αν δωθουν οι αριθμοι:
5 0 1 6 2 4 6 8 9 0 5 6 7 4 5 6 3 7
Να ενφανισθει το κατωθι ιστογραμμα:

```
  Συχνοτης
  -------------
0 |**
1 |*
2 |*
3 |*
4 |**
5 |***
6 |****
7 |**
8 |*
9 |*
```

Λυση

```
#include <iostream.h>
#define MAXEL 10   /* πληθος κελιων στον πινακα */
int main()
{
    int a[MAXEL],i,j;

    for(i=0; i <= MAXEL-1; i++)
        a[i] = 0; // Μιδενισμος πινακα

    while (1) {
        cin >> i;
        if (i<0)
            break;  // εαν αρνητικος τελος διαβασματος
        if (i > MAXEL-1) {     // εαν > 9 λαθος αριθμος
            cout << "Lathos\n";
            continue;
        }
        ++a[i]; // ιδιο με: a[i] = a[i] + 1;
    }
    // Εμφανιση αποτελεσματων
    cout << "  Συχνοτης" << endl;
    cout << "  -------------\n";
    for(i = 0; i <= MAXEL-1; i++) {  // Να περασουν ολα τα
                                     // κελια  του πινακα
        cout << i << " |";

        // τυπωσε τοσα (*) οσος ηταν ο μετρητης
        for(j = 1; j <= a[i]; j++)
            cout << '*';
        cout << endl;
    }
    return(0);
}
// p22.cpp //
```

Παραλλαγη: να βαλετε ελεγχο για την μεγιστη συχνοτητα των αριθμων (πχ. δεν επιτρετεται σε κανεναν αριθμο να δωθει περισσοτερες φορες απο 40)

-- Να γραφει ενα προγραμμα για το οποιο ισχυει οτι εις την προηγουμενη ασκηση αλλα το ιστογραμμα να εμφανιζεται καθετα δηλ.

```
 |
 |            *
 |          * *
 |*       * * * *
 |* * * * * * * * *
  --------------------
  0 1 2 3 4 5 6 7 8 9
```

Λυση

```
#include <iostream.h>
#define MAXEL 10         /* πληθος κελιων στον πινακα */
#define MAXFR  6         /* μεγιστη συχνοτητα */
int main()
{
    int a[MAXEL],i,j;
    for(i = 0; i <= MAXEL-1; i++) a[i] = 0;

    while (1) {
        cin >> i;
        if (i<0) break;
        if (i > MAXEL-1) {
            cout << "Lathos" << endl;
            continue;
        }
        ++a[i];
    }

    for(i = MAXFR; i >= 1; i--) {
        cout << "  |";
        for(j = 0; j <= MAXEL-1; j++) {
            if(i <= a[j])
                cout << "* ";
            else
                cout << "  ";
        }
        cout << "\n";
    }
    cout << "  --------------------\n   ";
    for(j = 0; j <= 9; j++)
        cout << j << " ";

    return(0);
}
// p23.cpp //
```

-- Μια μεθοδος για να υπολογισθουν ολοι οι πρωτοι αριθμοι (ενας αριθμος ονομαζεται πρωτος οταν διαιρειται μονο με τον εαυτό του και με το 1, με ολους τους

αλλους αριθμους οταν διαιτειται το πιλικον δεν ειναι ακεραιος αριθμος) που ειναι μικροτεροι ενος δοθεντος αριθμου n ειναι η επομενη:
α) θεωρησετε οτι ολοι οι αριθμοι 2,3,4,5,...,n ειναι πρωτοι.
β) διαγραψετε ολους τους αριθμους που ειναι πολλαπλασιοι του 2.
γ) στην συνεχεια διαγραψετε ολους τους αριθμους που ειναι πολλαπλασιοι του 3.
δ) συνεχισετε να διαγραφετε ολους τους αριθμους που ειναι πολλαπλασιοι του επομενου αριθμου μεχρι να φθασετε την τιμη που ειναι μεγαλυτερη απο την τετραγωνικη ριζα του n (πχ. αν n = 100 τοτε imax = 10)
ε) οποιος αριθμος απεμεινε ειναι πρωτος.
Γραψετε ενα προγραμμα το οποιο υπολογιζει τους πρωτους αριθμους μεχρι το 500. Χρησιμοποιησετε πινακα a[500+1].

```
#include <iostream.h>
#include <math.h>
#define MAXL      500
#define PRIME     1
#define NOT_PRIME 0

int main()
{
    int i,j,fmax,a[MAXL+1];
    // Ολοι οι αριθμοι θεωρουνται οτι ειναι πρωτοι
    for(i = 1; i <= MAXL; i++)
        a[i] = PRIME;
    fmax = (int) sqrt(MAXL);
    i = 2;
    while(i <= fmax) {
        // Οι πολλαπλασιοι αριθμοι δεν ειναι πρωτοι
        for(j = 2*i; j <= MAXL; j += i)
            a[j] = NOT_PRIME;
        i++; // ο επομενος
        while(a[i] == NOT_PRIME)
            i++; // παρεκαμψε ολους τους μη πρωτους
    }
    cout << "Πρωτοι αριθμοι μεχρι το " << MAXL << endl;

    for(i = 1; i <= MAXL; i++) {
        if (a[i] == PRIME)  // Εαν ειναι πρωτος εμφανισε τον
            cout << i << " ";
    }
    return(0);
}
// p24.cpp //
```

Συναρτησεις (Functions)

Οταν καποιος εργαζεται στην λυση ενος μεγαλου προβληματος και το προγραμμα αρχιζει και γινεται μεγαλο (μια σελιδα κωδικα ή και περισσοτερες) τοτε ειναι σημαντικο το προγραμμα να ειναι καλογραμμενο (καλοκατασκευασμενο). Για να ειναι ενα προγραμμα καλογραμμενο χρειαζεται να "σπασει" σε πολλα υποπρογραμματα. Ενα υποπρογραμμα ειναι στην ουσια ενα χωριστο προγραμμα με δικες του σταθερες, μεταβλητες κλπ.
Παραδειγματα συναρτησεων ειναι η **printf**, **scanf**, **gerchar**, κλπ. που παρεχονται απο την standard βιβλιοθηκη.

Ποτε πρεπει να δημιουργειται μια συναρτηση:
α) οταν ο κωδικας επαναλαμβανεται.
β) οταν το προβλημα απο μονο του περιεχει χωριστα υποπροβληματα.
γ) οταν ενα υποπροβλημα ειναι γενικης φυσης πχ. αναζητηση σε εναν πινακα, ταξινομηση ή παρουσιαση δεδομενων.

Ορισμος (πρωτοτυπο - prototype) συναρτησης:

```
τυπος_συναρτησης ονομα_συναρτησης(παραμετροι_εαν_υπαρχουν);
```

Παραδειγματα:

α) `void foo();` ιδιο με `void foo(void);`
Τυπος συναρτησης **void**
Ονομα συναρτησης **foo**
Καμμια παραμετρο

β) `int prn(int x);`
Τυπος συναρτησης **int**
Ονομα συναρτησης **prn**
Μια παραμετρο **int**

γ) `double wow(int i, char c, double f);`
Τυπος συναρτησης **double**
Ονομα συναρτησης **wow**
Τρεις παραμετρους **int**, **char**, **double**

Κληση συναρτησης (function call):

```
ονομα_συναρτησης(σταθερα ή μεταβλητη ή παρασταση)
```

Παραδειγματα

```
α) foo();
β) 1) k = prn(31);
   2) k = prn(a);
   3) g = prn(i+j);
```

Υλοποιηση συναρτησης (function)

```
τυπος_συναρτησης ονομα_συναρτησης(παραμετροι_εαν_υπαρχουν)
{
    // σωμα συναρτησης (εντολες)
}
```

Παραδειγματα

```
α) void foo()
   {
       cout << "Μεγας Αλεξανδρος" << emdl;
```

```
   }
β) int prn(int x)
   {
       int z;
       z = x * x;
       cout << z << endl;
       return( z );
   }
---
```

Η εντολη **return** χρησιμοποιειται οταν η συναρτηση επιστρεφει τιμη. Μια συναρτηση μπορει να εχει πολλες εντολες **return** αλλα επιστρεφει μονο μια τιμη.

Μια συναρτηση ειναι τυπου **void** οταν δεν επιστρεφει καμμια τιμη.

Οι παραμετροι (εαν υπαρχουν) σε μια συναρτηση πρεπει να δηλωνονται (τυπος και ονομα για καθε μια παραμετρο)

```
long foo(int x, int y, char c)   // Επικεφαλιδα συναρτησης
{
    // σωμα συναρτησης
}
```

- Το ονομα της συναρτησης ειναι **foo**.
- Η συναρτηση ειναι τυπου **long** διοτι επιστρεφει εναν **long** αριθμο.
- Η συναρτηση εχει τρεις παραμετρους εναν ακεραιο (**int x**) εναν ακομη ακεραιο (**int y**) και εναν χαρακτηρα (**char c**)

Παραδειγμα

```
#include <iostream.h>

void print(int);   // Προτωτυπο συναρτησης

int main()
{
    print(24);     // Κληση συναρτησης με σταθερα

    int a = 15;
    print(a);      // Κληση συναρτησης με μεταβλητη
    return(0);
}

// Υλοποιηση συναρτησης
void print(int x)       // Επικεφαλιδα συναρτησης
{
    cout << x << endl;  // Σωμα (εντολες) συναρτησης
}
```

Τροποι κλησεως συναρτησης

1. Με τιμη (call by value)

- Οι παραμετροι στην συναρτηση ειναι μεταβλητες.
- Η κληση της συναρτησης γινεται με μεταβλητες ή σταθερες ή παραστασεις.
- Η συναρτηση επεξεργαζεται αντιγραφα των μεταβλητων-κλησης.

Παραδειγμα

```
#include <iostream.h>
void disp(int);
int main()
{
    int k = 5;
    cout << "Πριν την κληση κ = " << k << endl;
    disp(k);  // Κληση: Με μεταβλητη
    /* Αλλοι τροποι κλησης. Το αποτελεσμα ειναι το ιδιο
       disp(5);      // Με σταθερα
       disp(k+2-2);  // Με παρασταση
    */
    cout << "Μετα την κληση κ = " << k << endl;
}

void disp(int k)
{
    k = k + 3;
    cout << "Μεσα στην συναρτηση κ = " << k << endl;
}
```

Αποτελεσμα

```
Πριν την κληση κ = 5
Μεσα στην συναρτηση κ = 8
Μετα την κληση κ = 5
```

2. Με δεικτη (call by reference)

- Οι παραμετροι στην συναρτηση ειναι δεικτες (pointers).
- Η κληση της συναρτησης γινεται με διευθυνσεις των μεταβλητων ή με δεικτες στις μεταβλητες
- Η συναρτηση επεξεργαζεται τις μεταβλητες-κλησης (εμεσως, μεσω δεικτων).

Παραδειγμα

```
#include <iostream.h>
void disp(int *);
int main()
{
    int k = 5;
    cout << "Πριν την κληση κ = " << k << endl;
    disp(&k);  // Κληση: Με διευθυνση
    /* Αλλος τροπος κλησης: Με δεικτη  Το αποτελεσμα ειναι το ιδιο
       int *ptr;
       ptr = &k;
       disp(ptr);
    */
    cout << "Μετα την κληση κ = " << k << endl;
}
```

```
void disp(int *k)
{
    *k = *k + 3;
    cout << "Μεσα στην συναρτηση κ = " << k << endl;
}
```

Αποτελεσμα

```
Πριν την κληση κ = 5
Μεσα στην συναρτηση κ = 8
Μετα την κληση κ = 8
```

3. Με αναφορα (call by reference)

- Οι παραμετροι στην συναρτηση ειναι αναφορες (references).
- Η κληση της συναρτησης γινεται με μεταβλητες ή με αναφορες
- Η συναρτηση επεξεργαζεται τις μεταβλητες-κλησης (εμεσως, μεσω αναφορων).

Παραδειγμα

```
#include <iostream.h>
void disp(int &);
int main()
{
    int k = 5;
    cout << "Πριν την κληση κ = " << k << endl;
    disp(k);  // Κληση: Με μεταβλητη
    /* Αλλος τροπος κλησης: Με αναφορα  Το αποτελεσμα ειναι το ιδιο
       int &r = k;
       disp(r);
    */
    cout << "Μετα την κληση κ = " << k << endl;
}

void disp(int &k)
{
    k = k + 3;
    cout << "Μεσα στην συναρτηση κ = " << k << endl;
}
```

Αποτελεσμα

```
Πριν την κληση κ = 5
Μεσα στην συναρτηση κ = 8
Μετα την κληση κ = 8
```

Υπερφορτωση συναρτησεων

Υπερφορτωση συναρτησης (function overloading) εχουμε οταν χρησιμοποιουμε το ιδιο ονομα σε δυο ή περισσοτερες συναρτησεις. Οι συναρτησεις που εχουν το ιδιο ονομα πρεπει να διαφερουν το λιγοτερο σε μια παραμετρο:

α) Διαφορετικο αριθμο παραμετρων

```
int  foo(int,int,int);      // Πρωτοτυπα συναρτησεων
int  foo(int);
int  foo();
```

β) Διαφορετικο τυπο παραμετρων

```
int  boo(int,float);        // Πρωτοτυπα συναρτησεων
int  boo(float,int);
```

```
int  boo(int,int);
```

γ) Συνδιασμο των παραπανω α, β

```
int   zoo(int,int);        // Πρωτοτυπα συναρτησεων
char  zoo(char,int,float);
float zoo(int);
```

Παραδειγμα

```
#include <iostream.h>

void print(int);   // Υπερφορτωση συναρτησης print
void print(long);

int main()
{
    print(24);     // Κληση συναρτησης με int
    print(23456L); // Κληση με long

    return(0);
}

void print(int x)       // Επικεφαλιδα συναρτησης
{
    cout << x << endl;
}

void print(long x)       // Επικεφαλιδα συναρτησης
{
    cout << x << endl;
}
```

Αρχικες τιμες σε παραμετρους συναρτησεων

Η C++ επιτρεπει να δωσουμε αρχικη τιμη (default) σε παραμετρο συναρτησης. Εαν κατα την κληση της συναρτησης παραλειπεται το ορισμα τοτε στην συναρτηση χρησιμοποιειται η αρχικη τιμη που αντιστοιχει στην παραμετρο.

α) `void fn(double d = 2.8)` // Αρχικη (εξ'ορισμου) τιμη στο `d = 2.8`

```
{
   // ...
}
```

Κληση συναρτησης

`fn(23.567);` // Το `d` λαμβανει την τιμη `23.567`

ή

`fn();` // Η συναρτηση χρησιμοποιει για το `d` την αρχικη τιμη `2.8`

β)

```
int boo(char *s,long k=0, char t='\n')
{
   // ...
}
```

Κληση

```
a = boo("Lucky day");       // Αρχικες τιμες k=0 και t='\n'
a = boo("Hope", 345L);      // Αρχικη τιμη t='\n'
a = boo("Light", 12L,'\t');
```

Απαξ' και ορισθει μια παραμετρος με αρχικη τιμη, ολες οι υπολοιπες παραμετροι πρεπει να εχουν αρχικη τιμη.

```
// int hoo( int i=10, int j);// ΛΑΘΟΣ.
```
Και η παραμετρος `j` πρεπει να εχει αρχικη τιμη

Η προηγουμενη συναρτηση θα μπορουσε να γραφει:

```
int hoo(int i=10, int j=17); // Σωστη
```
ή
```
int hoo(int i, int j=17);    // Σωστη
```

Ασκησεις

-- Να γραφει μια συναρτηση abs η οποιο επιστρεφει την απολυτο τιμη μιας ακεραιας μεταβλητης.

Λυση

```
#include <iostream.h>
int abs(int);  // Πρωτοτυπο συναρτησης
int main()
{
    int x,y;

    x = -123;
    y = abs(x); // Κληση της συναρτησης
    cout << "Απολυτη τιμη του " << x << " ειναι " << y
         << endl;
    return(0);
}

// Υλοποιηση της συναρτησης
int abs(int x)  // Επικεφαλιδα συναρτησης
{
    if (x < 0)
        x = -x;
    return(x); // Η συναρτηση επιστρεφει εναν αριθμο
               // τυπου int
}
```

2ος τροπος υλοποιησης συναρτησης

```
int abs(int x)
{
   return(x < 0 ? -x : x);
}
```

3ος τροπος υλοποιησης: η συναρτηση ως Macro

```
#define ABS(x) (x < 0 ? -x : x)
```

Παραδειγμα με τον 3ο τροπο

```
#include <iostream.h>
#define ABS(x) (x < 0 ? -x : x)
int main()
{
    cout << "Απολυτος τιμη του -1: " << ABS(-1) << endl;
    cout << "Απολυτος τιμη του  1: " << ABS(1) << endl;
    return(0);
}
```

-- Να γραφει μια συναρτηση clear() που καθαριζει την οθονη του τερματικου.

```
#include <stdlib.h>
void clear();
int main()
{
    clear();   // Καλειται να εκτελεσθει η συναρτησης.
    return(0);
}
void clear()
{
    system("clear");  // Στο DOS χρησιμοποιεισετε: system("cls");
}
```

-- Να γραφει μια συναρτηση int ctoi η οποια επιστρεφει για εναν αριθμητικο χαρακτηρα τον αντιστοιχο ακεραιο (πχ. αν ο αριθμητικος χαρακτηρας ειναι ο '6' τοτε η συναρτηση θα επιστρεψει τον ακεραιο 6). Εαν ο χαρακτηρας δεν ειναι αριθμητικος τοτε συναρτηση επιστρεφει -1.

```
int ctoi(char ch)
{
    int a = -1;
    if (ch >= '0' && ch <= '9')
          a = ch - '0';
    return(a);
}
```

-- Να γραφει μια συναρτηση char itoc η οποια μετατρεπει εναν ακεραιο σε χαρακτηρα. Εαν ο αριθμος δεν βρισκεται στην περιοχη 0,1,...8,9 τοτε συναρτηση επιστρεφει τον χαρακτηρα null '\0'.

-- Να γραφει μια συναρτηση digit η οποια ελεγχει εναν χαρακτηρα και επιστρεφει 1 εαν ο χαρακτηρας ειναι αριθμητικος και 0 εαν δεν ειναι αριθμητικος.

Λυση

```
#include <iostream.h>
int digit(char);
int main()
{
    char c;


    c = '6';
    if (digit(c))
        cout << "Ο χαρακτηρας " << c <<
```

```
                " ειναι αριθμητικος\n";
    else
        cout << "Ο χαρακτηρας " << c <<
                " ΔΕΝ ειναι αριθμητικος\n";
    return(0);
}

int digit(char ch)
{
    int r;
    if (ch >= '0' && ch <= '9')
        r = 1;
    else
        r = 0;
    return(r);
}
```

Αλλος τροπος υλοποιησης συναρτησης:

```
int digit(char ch)
{
    return(ch >= '0' && ch <= '9' ? 1 : 0);
}
```

-- Παραδειγμα κλησης συναρτησης με τιμη (call by value) Να γραφει μια συναρτηση swap η οποια αλλαζει αμοιβαια τις τιμες δυο μεταβλητων x και y.

```
#include <iostream.h>
void swap(int,int);
int main()
{
    int x,y;
    x = 5;
    y = 3;
    cout << "Πριν την εκτελεση της συναρτησης x=" << x
         << " y=" << y << endl;
    swap(x,y);
    cout << "Μετα την εκτελεση της συναρτησης x=" << x
         << " y=" << y << endl;
    return(0);
}

void swap(int x,int y)
{
   int t;
   t = x;
   x = y;
   y = t;
   cout << "Μεσα στην συναρτηση ..... x=" << x
        << " y=" << y << endl;
}
// p25.cpp //
```

Αποτελεσμα (δεν εγινε τελικα η αμιβαια αλλαγη):

```
Πριν την εκτελεση της συναρτησης x=5, y=3
Μεσα στην συναρτηση ..... x=3, y=5
Μετα την εκτελεση της συναρτησης x=5, y=3
```

-- Παραδειγμα κλησης συναρτησης με δεικτες (call by reference) Να γραφει μια συναρτηση swap η οποια αλλαζει αμοιβαια τις τιμες δυο μεταβλητων x, και y.

```
#include <iostream.h>
void swap(int *,int *);
int main()
{
    int x,y;

    x = 5;
    y = 3;
    cout << "Πριν την εκτελεση της συναρτησης x=" << x
         << " y=" << y << endl;
    swap(&x,&y); // Κληση της συναρτησης με διευθυνσεις των
                 // μεταβλητων

    cout << "Μετα την εκτελεση της συναρτησης x=" << x
         << " y=" << y << endl;
    return(0);
}

void swap(int *x, int *y)   // Οι παραμετροι ειναι δεικτες
{
   int t;
   t = *x;
   *x = *y;
   *y = t;
   cout << "Μεσα στην συναρτηση ..... x=" << *x
        << " y=" << *y << endl;
}
// p26.cpp //
```

Αποτελεσμα

```
Πριν την εκτελεση της συναρτησης x=5, y=3
Μεσα στην συναρτηση ..... x=3, y=5
Μετα την εκτελεση της συναρτησης x=3, y=5
```

-- Να γραφει μια συναρτηση max η οποια ελεγχει δυο αριθμους και επιστρεφει τον μεγαλυτερο εκ των δυο αριθμων.

```
int max(int x, int y)
{
    return(x > y ? x : y);
}
```

-- Να γραφει μια συναρτηση max3 η οποια ελεγχει τρεις αριθμους και επιστρεφει τον μεγαλυτερο εκ των τριων αριθμων.

```
#include <iostream.h>

float max3(float x,float y,float z);

int main()
```

```
{
    float m,max3();

    mx = max3(450.0, 300.0, 60.0);
    cout << "Max=" << mx << endl;
    return(0);
}

float max3(float x,float y,float z)
{
    float m;
    if (x > y)  m = x; else m = y;
    if (z > m)  m = z;
    return(m);
}
```

Εναλλακτικη λυση

```
float max3(float x,float y,float z)
{
    return(x > y ? (x > z ? x : z) : (y > z ? y : z));
}
```

-- Να γραψετε μια συναρτηση odd που ελεγχει εναν αριθμο εαν ο αριθμος ειναι ζυγος τοτε επιτρεφει 1 εαν ειναι μονος τοτε επιστρεφει 0.

```
#include <iostream.h>
#define TRUE   1
#define FALSE  0
int odd(int);
int main()
{
    int r,i;
    i = 14;
    r = odd(i);
    cout << "Ο αριθμος " << i << " ειναι " <<
            (r == 1 ? "ΖΥΓΟΣ" : "μονος");
    return(0);
}

int odd(int x)
{
    if(x/2 == x/2.0) return(TRUE);
    else return(FALSE);
}
```

-- Να γραψετε μια συναρτηση dev με δυο float παραμετρους x και y, και επιστρεφει 1 εαν η τιμη x ειναι διαιρετεα με το y, αλλιως επιστρεφει 0.

-- Γραψετε μια συναρτηση area() που υπολογιζει το εμβαδον ενος τριγωνου οταν οι τρεις πλευρες x, y και z ειναι γνωστες. Το εμβαδον υπολογιζεται με την φορμα:

εμβαδον = $\sqrt{p(p-x)(p-y)(p-z)}$

οπου $p = (x+y+z)/2$

Χρησιμοποιεισετε **#include <math.h>** για να κανετε χρηση της συναρτησης **sqrt()** της βιβλιοθηκης της C/C++.

-- Να γραφει καταλληλος κωδικας (δυο συναρτησεις abs) οπου υπολογιζεται και επιστρεφεται η απολυτη τιμη μιας ακεραιας ή πραγματικης μεταβλητης. Να χρησιμοποιειθει υπερφωρτωση συναρτησεων.

```
#include <iostream.h>
int abs(int);      // Ονομα συναρτησης ιδιο αλλα διαφορετικοι
double abs(double);// τυποι παραμετρων.
int main()
{
    int i; double d;
    cin >> i;
    cin >> d;
    i = abs(i);
    d = abs(d);
    cout << "Απολυτη τιμη του ακερ.  i=" << i << endl;
    cout << "Απολυτη τιμη του πραγμ. d=" << d << endl;
    return(0);
}

int abs(int x)
{
    return( x < 0 ? -x : x );
}

double abs(double y)
{
    return( y < 0 ? -y : y );
}
```

-- Να γραφουν δυο συναρτησεις toint(*παραμετρος1*) οπου μετατρεπεται και επιστρεφεται η *παραμετρος1* ως ακεραιος. Η *παραμετρος1* μπορει να ειναι πραγματικος ή χαρακτηρας. Να χρησιμοποιειθει υπερφωρτωση συναρτησεων.

-- Να γραφουν δυο συναρτησεις embadon οπου υπολογιζεται το εμβαδον ενος τετραγωνου (κληση με νια μονο παραμετρο) ή το εμβαδον ενος ορθογωνιου (κληση με δυο παραμετρους). Να χρησιμοποιειθει υπερφωρτωση συναρτησεων.

Προεπεξεργαστης (Preprocessor)

Η μεταφραση ενος πηγαιου προγραμματος γινεται σε πολλες φασεις. Μια απο τις φασεις ειναι η προεπεξεργρασια. Σε αυτην την φαση ο compiler επεξεργαζεται τις οδηγιες που προοριζονται για τον εαυτον του. Οι οδηγιες για τον compiler σε ενα πηγαιο προγραμμα ονομαζονται οδηγιες προεπεξεργαστη (processor directives).

Ο προεπεξεργαστης χειριζεται τις ακολουθες οδηγιες.

#if	Εαν
#ifdef	Εαν εχει ορισθει
#ifndef	Εαν δεν εχει ορισθει
#else	Διαφορετικα
#elif	Διαφορετικα Εαν
#endif	Τελος του Εαν
#include	Να συμπεριλαβεις
#define	Ορισμος
#undef	Ακυρωση Ορισμου
#pragma	Οδηγιες προς στον μεταφραστη

Με την οδηγια **#define** οριζονται σταθερες, εντολες και macros:
Ορισμος σταθερας

```
#define PI 3.1415
```

Ορισμος εντολης

```
#include <stdio.h>
#define MES printf("Winter song\n")  // εντολη
main()
{
    MES;
}
```

Παραδειγματα

```
a) #include <stdio.h>    // Αρχειο επικεφαλιδων του μεταφραστη
   #include "myfile.h"   // Αρχειο επικεφαλιδων του χρηστη
```

```
b) #define TRUE  1    // Ορισε
   #define FALSE 0    // Ορισε
   :
   int x = 1;
   :
   if( x == TRUE)
       cout << "Res = " << TRUE << " " << TRUE+1 << endl;

c) #define LATHOS "Λαθος αριθμος στην εισοδο\n"
   :
   cout << LATHOS;

d) #define MAX 100
   :
   double ar[MAX];
   :
   for(i = 0; i < MAX; i++)
       cout << ar[i];
e) #define MAX        // Ορισμος του MAX
   :
   double ar[MAX];
   :
   #undef MAX         // Ακυρωση του MAX
   // Σε αυτο το σημειο το MAX ειναι αοριστο (ακυρωθηκε)

f) #ifndef EOL         // Εαν δεν ειναι ορισμενο το EOL
   #define EOL '\n'    // Ορισε το EOL
   #endif              // Τελος του #if

g) #define VisualC  0
   #define TurboC   1
   #define DecC     2
   :
   #define METAFRASTHS VisualC
   :
   #if METAFRASTHS == VisualC
       char etaireia[] = "MicroSoft";
   #elif METAFRASTHS == TurboC
       char etaireia[] = "Borland";
   #else
       char etaireia[] = "Digital Equipment";
   #endif
```

Συναρτησεις ως Macros

Η οδηγια **#define** μπορει να παρει και παραμετρους ετσι ειναι δυνατον να ορισουμε macros. Σε οποιοδηποτε σημειο του πηγαιου προγραμματος γινεται χρηση της macro το ονομα της macro αντικαθισταται με τον αντιστοιχο κωδικα. Ετσι το προγραμμα κερδιζει σε ταχυτητα (η κληση και εκτελεση συναρτησης ειναι πιο αργη απο τον ιδιο το κωδικα) αλλα μεγαλωνει σε μεγεθος (ο κωδικας της macro επαναλαμβανεται)
Εαν εχουμε τον ακολουθο πηγαιος κωδικας

```
#define MAX2(x,y) (x > y ? x : y)
```

```
:
int i,j,z;
:
z = MAX2(i,j);
```

Μετα την προεπεξεργασια το πηγαιο προγραμμα μετατρεπεται σε:

```
int i,j,z;
:
z = (i > j ? i : j);
```

Δηλαδη, Αντικαθισταται το **MAX2** απο τον αντιστοιχο κωδικα.

-- Παραδειγμα. Η macro TOUPPER μετατρεπει ενα πεζο γραμμα (χαρακτηρα) σε κεφαλαιο γραμμα (χαρακτηρα)

```
#include <iostream.h>

// Ορισμος της macro
#define TOUPPER(c) (c >= 'a' && c <= 'z' ? c - 32 : c)


int main()
{
    char x;
    cin >> x;
    cout << TOUPPER(x) << endl;
    return(0);
}
```

inline συναρτησεις

Οι **inline** συναρτησεις ειναι macros αλλά δηλωνονται και υλοποιουνται οπως οι "κανονικες" συναρτησεις με προθεμα την λεξη inline.

-- Παραδειγμα. Η inline συναρτηση TOUPPER μετατρεπει εναν πεζο γραμμα (χαρακτηρα) σε κεφαλαιο γραμμα (χαρακτηρα)

```
#include <iostream.h>

inline char TOUPPER(char c)
{
    if(c >= 'a' && c <= 'z')
        c = c - 32;
    return(c);
}

int main(void)
{
    char x;
    cin >> x;
    cout << TOUPPER(x) << endl;
    return(0);
}
// p72.cpp //
```

Αναφορες (References)

Μια αναφορα ειναι ενας ειδικος "δεικτης" που δημιουργει ενα επιπλεον ονομα σε ενα αντικειμενο (μεταβλητη, δομη κοκ). Οι αναφορες ειναι χρησιμες στην κληση συναρτησεων.
Ο τελεστης αναφορας ειναι ο **&**.

```
int i;
int &r = i; // Ο r ειναι "συνονυμο" του i.
            // Ο r πρεπει να παρει τιμη κατα την δηλωση.
```

-- Να γραφει μια συναρτηση void swap η οποια αλλαζει αμοιβαια τις τιμες δυο μεταβλητων x, και y κανοντας χρηση των αναφορων (εχουμε κληση με αναφορα call by reference)

```
#include <iostream.h>
void swap(int&, int&);  // Δηλωση πρωτοπυπου συναρτησης
int main()
{
   int x,y;
   x = 5;
   y = 3;
   cout << "Πριν την εκτελεση της συναρτησης x=" << x
        << " y=" << y << endl;
   swap(x,y); // Κληση της συναρτησης με τις μεταβλητες
              // Δεν χρειαζεται πλεον ο τελεστης &.
   cout << "Μετα την εκτελεση της συναρτησης x=" << x
        << " y=" << y << endl;
   return(0);
}

void swap(int& a, int& b)  // Το a ειναι b αναφορες
{
   int t;
   t = a;    // Οι αναφορες δεν χρειαζονται το τελεστη *
   a = b;
   b = t;
```

```
   cout << "Μεσα στην συναρτηση ..... x=" << a
        << " y=" << b << endl;
}
// p27.cpp //
```

Ως αποτελεσμα εχουμε την αλλαγη των τιμων.

-- Να γραφει η συναρτηση void absx(int&, long&) η οποια αλλαζει τις τιμες ενος ακεραιου και ενος μεγαλου ακεραιου κανοντας χρηση αναφορων. (Εαν οι τιμες ειναι αρνητικες τοτε γινονται θετικες αλλιως παραμενου ως εχουν)

```
#include <iostream.h>
void absx(int &,long &);
int main()
}
    int i;
    long k;
    i = -10;
    k = -56L;

    cout << "Τιμες πριν την κληση " << i << ", " << k
         << endl;
    absx(i,k);
    cout << "Τιμες μετα την κληση " << i << ", " << k
         << endl;
    return 0;
}

void absx(int &x,long &y)
{
    if(x < 0) x = -x;
    if(y < 0) y = -y;
}
```

Αποτελεσμα

```
Τιμες πριν την κληση -10, -56
Τιμες μετα την κληση  10,  56
```

-- Να γραφει μια συναρτηση void ctou(char &x, int& flag) η οποια ελεγχει εναν χαρακτηρα x εαν ειναι ειναι πεζος να τον μετατρεπει σε κεφαλαιο και η μεταβλητη flag γινεται 1, αλλιως ο χαρακτηρας παραμενει ως εχει και η μεταβλητη flag γινεται 0. Στο κυριως προγραμμα να εμφανισθει ο χαρακτηρας και αντιστοιχο μηνυμα για το αν χαρακτηρας μετετραπει σε κεφαλαιο.

```
#include <iostream.h>
void rtoup(char &, int &);

int main()
{
    int fl;
    char ch;
    cout << " Δωσε εναν χαρακτηρα ";
    cin >> ch;

    rtoup(ch,fl);
    cout << "Ο χαρ. ειναι " << ch << endl;
```

```
    if ( fl == 1 )
        cout << "Εγινε μετατροπη" << endl;
    else
        cout << "ΔΕΝ εγινε μετατροπη" << endl;
    return 0;
}

void rtoup(char &x, int& flag)
{
    flag = 0;
    if (x >= 'a' && x <= 'z') {
        x = x - 32;
        flag = 1;
    }
}
// p36.cpp //
```

-- Να γραφει μια συναρτηση η οποια επιστρεφει την αναφορα ενος κελιου πινακα ακεραιων (Το στοιχειο a[k] θα εχει συνονυμο το a(k)).

```
#include <iostream.h>
int& arr(int);  // Πρωτοτυπο συναρτησης
int ar[10];
int main()
{
    int i;

    for ( i = 0; i <= 9; i++)
        arr(i) = i*i;          // Η αναφορα παιρνει τιμη i*i

    // Εμφανηση του πινακα
    for ( i = 0; i <= 9; i++)
        cout << arr(i) << " ";

    int sum = 0;
    for ( i = 0; i <= 9; i++)
        sum = sum + arr(i);

    cout << "Αθροισμα πινακα = " << sum << endl;
    return 0;
}

int& arr(int k)
{
    return ar[k];
}
```

Πινακες ως παραμετροι σε συναρτησεις

(Το ονομα ενος πινακα ειναι η διευθυνση του πρωτου στοιχειου του πινακα)

Το περασμα **μονοδιαστατου** πινακα ως παραμετρο σε συναρτηση γινεται ως εξης:
A. Με χρηση αγκυλων `[]`

```
void func(int []);  // Στο πρωτοπυπο: Τυπος πινακα και Αγκυλες
       :
    int b[100];     // Δηλωση πινακα
       :
    func(b);        // Στην κληση: Το ονομα του πινακα
       :
       :
void func(int array[]) // Στην επικεφαλιδα: Τυπος, Ονομα, και Αγκυλες
{
    // Σωμα συναρτησης
}
```

Στην επικεφαλιδα της συναρτησης δεν αναφερονται το πληθος κελλιων του πινακα.

B. Με χρηση δεικτη *

```
int www(char *);    // Στο πρωτοπυπο: Τον τυπο του δεικτη
    :
    char s[80]; int y;
    :
    y = www(s);     // Στην κληση: Το ονομα του πινακα
    :
    :
int www(char *ptr) // Στην επικεφαλιδα:  τον τυπο και το ονομα του δεικτη
{
    ...
}
---
```

Το περασμα πινακα **δυο διαστασεων** ως παραμετρο σε συναρτηση γινεται ως εξης: Το ονομα του πινακα με τις αγκυλες χωρις να γραφονται οι γραμμες του πινακα αλλα να γραφονται οι στηλες του πινακα.
Παραδειγματα

α)
```
void func(int [][100]);        // Πρωτοπυπο
:
int arr[4][100];
:
func(arr);                     // Κληση
:
:
// Δηλωνεται το πληθος των στηλων (100)
void func(int arr2d[][100])  // Επικεφαλιδα
{
    ...
}
```
Η συναρτηση μπορει να επεξεργαστει πινακες ακεραιων δυο διαστασεων μεχρι 100 στηλες.

β) Πινακας χαρακτηρων 2 διαστασεων. Στηλες 30.
```
double foo(char w[][30],int r)
{
    ...
}
```
Η συναρτηση μπορει να επεξεργαστει πινακες χαρακτηρων δυο διαστασεων μεχρι 30 στηλες.

Ασκησεις

-- Να γραφει μια συναρτηση zero_array η οποια μηδενιζει εναν πινακα ακεραιων αριθμων με n στοιχεια.

```
void zero_array(int [], int); // Πρωτοτυπο

int main()
{
    int a[50];
    zero_array(a,50);
}

void zero_array(int ar[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        ar[i] = 0;
}
```

-- Να γραφει μια συναρτηση zero_array2 η οποια μηδενιζει εναν πινακα ακεραιων αριθμων δυο διαστασεων με g γραμμες και k στηλες. Η συναρτηση να μπορει να χειρισθει πινακες με μεγιστο αριθμο στηλων 100.

```
void zero_array2(int ar[][100],int g,int k)
{
   int i,j;
   for (i = 0; i < g; i++)
       for (j = 0; j < k; j++)
          ar[i][j] = 0;
}
```

-- Να γραφει μια συναρτηση int sum_array η οποια επιστρεφει το αθροισμα των n πρωτων στοιχειων ενος πινακα ακεραιων αριθμων.

```
int sum_array(int ar[],int n)
{
```

```
    int i, s = 0;
    for (i = 0; i < n; i++)
        s += ar[i];
    return(s);
}
```

-- Να γραφει μια συναρτηση double average_array η οποια επιστρεφει την μεση τιμη n ακεραιων αριθμων ενος πινακα.

```
double average_array(int ar[],int n)
{
    int i;  double s = 0.0;
    for (i = 0; i < n; i++)
        s += ar[i];
    return(s/n);
}
```

-- Να γραφει μια συναρτηση η οποια εμφανιζει στην οθονη τις δυο διαγωνιες για πινακες με n γραμμες και n στηλες.

-- Να γραφει μια συναρτηση η οποια υπολογιζει την variance των τιμων ενος πινακα. Η ευσεση της variance γινεται ως εξης:

1. Αθροιζουμε τους αριθμους του πινακα

$$sum = \sum_{i=0}^{i=n} a[i]$$

2. Ευρεση μεσης τιμης των αριθμων

   `aver = sum / n`

3. Για καθε αριθμο αθροιζουμε το τετραγωνο της διαφορας (αριθμος - μεση_τιμη)

$$A = \sum_{i=0}^{i=n} (a[i] - aver)*(a[i] - aver)$$

4. variance = A (αθροισμα) δια του πληθους των αριθμων

   `variance = A / n`

Λυση

```
#define square(x)   (x*x)

double variance(int arr[],int n)
{
    double aver, sum = 0.0;
    int i;

    for (i = 0; i < n; i++)      // 1. ***
        sum = sum + arr[i];
    aver = sum / n;              // 2. ***
    sum = 0.0
    for (i = 0; i < n; i++)
        sum = sum + square((arr[i] - aver));  // 3. ***
    return( sum/n );                          // 4. ***
}
```

Ακολουθιες χαρακτηρων (Strings)

String ειναι μια ακολουθια απο χαρακτηρες. Στην C/C++ δεν υπαρχει τυπος δεδομενων string, αλλα τα string οριζονται ως πινακες απο χαρακτηρες. Εξ ορισμου στην C/C++ ο '\0' (null χαρακτηρας) σημειωνει (μαρκαρει) το τελος του περιεχομενου ενος string. Αυτο σημαινει οτι ενα string ειναι ενας πινακας χαρακτηρων που το τελευταιος χαρακτηρας ειναι ο '\0' (null terminated array)

Πινακες χαρακτηρων μιας διαστασης

Ορισμος πινακα χαρακτηρων (string):

```
char ονομα_πινακα[αριθμος_κελλιων];
```

Παραδειγμα

```
char s[30];  // Δηλωση πινακα 30 χαρακτηρων.
```

Το string s μηδενιζεται (κενο, αδειο) ως εξης:

```
s[0] = '\0'; // Το μηκος του string s ειναι 0
```

```
#include <stdio.h>
int main()
{
    char tex[5];      // πινακας χαρακτηρων 5-θεσεων
    tex[0] = 'A';
    tex[1] = 'd';
    tex[2] = 'a';
    tex[3] = 'm';
    tex[4] = '\0';    // τελευταιος χαρακτηρας
    cout << "|" << tex << "|" << endl;
    return(0);
}
```

Το αποτελεσμα εμφανιζεται ως μια λεξη:

```
|Adam|
```

Επειδη η C/C++ δεν εχει τελεστες για χειρισμο των string, οι compilers διαθετουν βιβλιοθηκη με συναρτησεις string, πχ. strcpy, strcat κλπ. Για να χρησιμοποιησουμε

τις string συναρτησεις πρεπει να συμπεριλαβουμε στο προγραμμα μας το **<string.h>** Οι string συναρτησεις της βιβλιοθηκης απαιτουν (ως επι το πλειστον) την παρουσια του τελευταιου null χαρακτηρα.

α) `cout << "Eva";` // Ο compiler εδω προσθετει ενα '\0'.
β)
```
char text[81];
:
strcpy(text,"com");     // Αντεγραψε 'c' , 'o' , 'm' , '\0' στο text.
strcat(text,"piler"); // Προσθεσε 'p' , 'i' , 'l' , 'e' , 'r' , '\0'
```
Τωρα το text περιεχει την λεξη **compiler**
Οι συναρτησεις strcpy, και strcat ανηκουν στην καθιερωμενη βιβλιοθηκη.

```
#include <iostream.h>
#include <string.h>
int main()
{
    // Εισαγωγη στρινγ απο πληκτρολογιο, εμφανιση στρινγ
    char nam[30];

    cout << "Γραψε το ονομα σου: ";
    cin >> nam;
    cout << "Μπραβο " << nam << endl;

    char s[50];
    // Αρχικη τιμη (σταθερα στρινγ) σε στρινγ
    strcpy(s, "Lets dance ");
    cout << s << endl;

    // Στο τελος του στρινγ s προστιθεται το στρινγ nam
    strcat(s,nam);
    cout << s << endl;

    return(0);
}
```

Αν ο χρηστης δωσει ως ονομα **Eva** τοτε θα εμφανισθουν τα παρακατω μηνυματα:
```
Μπραβο Eva
Lets dance
Lets dance Eva
```

Πινακες χαρακτηρων δυο διαστασεων

Οι πινακες χαρακτηρων δυο διαστασεων (πινακες string) δηλωνονται ως εξης:
char *ονομα_πινακα*[*αριθμος_γραμμων*][*αριθμος_στηλων*];

Εαν ο πινακας χρησιμοποιειθει να αποθηκευσει strings τοτε καθε γραμμη του πινακα πρεπει να τελειωνει με '\0' (πινακας απο strings).
α) `char user_names[10][20];`
Πινακας χαρακτηρων δυο διαστασεων (10 γραμμες, 20 στηλες)
ή πινακας απο 10 strings (καθε string μπορει να εχει μηκος 19+1 χαρακτηρες.

β) `char fbc[3][10] = { "PAOK", "ARIS", "IRAKLIS" };`

```
γ) char mines[][5] = { "ΙΑΝ", "ΦΕΒ", "ΜΑΡ", "ΑΠΡ",
                       "ΜΑΙ", "ΙΟΥΝ", "ΙΟΥΛ", "ΑΥΓ",
                       "ΣΕΠ", "ΟΚΤ", "ΝΟΕ", "ΔΕΚ" };

ε) char carr[4][11]; // Πινακας με 4 string μεχρι 10+1(null) χαρακτηρες
                     // καθε string.
   strcpy(carr[0],"C");
   strcpy(carr[1],"Pascal");
   strcpy(carr[2],"");
   strcpy(carr[3],"Object C++");
```

Ο πινακας carr καταλαμβανει στην μνημη 4x11=44 bytes. Παρ'ολα αυτα μονο 1+6+0+10=17 bytes χρησιμοποιουνται. Καθε string μπορει να εχει μηκος απο 1-10 (η κελι 11 ειναι για το '\0') (δεν πρεπει να υπερβαινει το μηκος 10)

Πινακες χαρακτηρων ως παραμετροι σε συναρτησεις

Το περασμα πινακων χαρακτηρων σε συναρτησεις ακολουθουν τους κανονες των πινακων αριθμων.

Μονοδιαστατοι πινακες:

α) Χρηση αγκυλων `[]`

```
char foo(char [], int);     // Πρωτοτυπο
:
char s[30];
:
ch = foo(s,30); // Ιδιο με: ch = foo(&s[0],30); // Κληση
:
char foo(char ar[], int n) // Επικεφαλιδα
{
    //
}
```

β) Χρηση δεικτη `*`

```
char boo(char *,int);       // Πρωτοτυπο
:
char m[50];
:
k = boo(m,50);              // Κληση
:
int boo(char *ar, int n)    // Επικεφαλιδα
{
   //
}
```

Διδιαστατοι Πινακες

Για τους διδιαστατους πινακες δινεται υποχρεωτικα ο αριθμος των στηλων (Με την χρηση δεικτων μπορουμε να ξεπερασουμε αυτην τον περιορισμο).

```
int zoo(char [][50], int);   // Πρωτοτυπο συναρτησης
```

```
:
char ar[4][50];
:
m = zoo(ar,4);              // Κληση
:
int zoo(char car[][50], int n) // Επικεφαλιδα
{
    //
}
```

Ασκησεις

-- Να γραφει ενα προγραμμα το οποιο διαβαζει χαρακτηρες απο το τερματικο και τους τοποθετει σε εναν πινακα χαρακτηρων. Τελος να εμφανισθουν οι χαρακτηρες και το πληθος των.

```
#include <iostream.h>
int main()
{
    char ch, a[41];
    int i = 0;
    while ( cin.get(ch) )
        a[i++] = ch;
    // Εαν ο πινακας a θεωρηθει ως string πρεπει
    // να τερματισθει δινοντας το a[i] = '\0';
    cout << "Πληθος χαρακτηρων: " << i << endl;
    int j;
    for(j = 0; j < i; j++)
        cout << a[j];
    return(0);
}
// p23a.cpp //
```

Παραλλαγη Να τροποποιηθει το προγραμμα ετσι ωστε να μη διαβαζει περισσοτερους απο 40 χαρακτηρες.

-- Να γραφει ενα προγραμμα το οποιο διαβαζει χαρακτηρες απο το τερματικο και τους τοποθετει σε εναν πινακα χαρακτηρων A (τελος διαβασματος χαρακτηρων με <Enter>) Στην συνεχεια αντιγραφει τον πινακα A σε εναν αλλο πινακα B. Τελος εμφανιζει τον πινακα A ως string και τον πινακα B χαρακτηρα προς χαρακτηρα.

```
#include <iostream.h>
int main()
{
    char a[61], b[61];
    int i = 0;
```

```
    // Διαβασμα χαρακτηρων στον πινακα
    while ( cin.get(ch) )
        a[i++] = ch;
    a[i] ='\0'; // Τερματισμος του πινακα

    // Αντιγραφη του πινακα a στον πινακα b
    i = 0;
    while (a[i] != '\0') {
        b[i] = a[i];
        i++;
    }
    b[i] = '\0';  // Τερματισμος του πινακα

    // Εμφανιση του πινακα a ως string
    cout << "Πινακας A: " << a << endl;

    // Εμφανιση του πινακα b χαρακτηρα προς χαρακτηρα
    cout << "Πινακας B: ";


    i = 0;
    while (b[i] != '\0') {
        cout << b[i] << " ";
        i++;
    }
    cout << "\n";

    return(0);
}
// p23b.cpp //
```

-- Να γραφει μια συναρτηση που αντιγραφει ενα string s στο string t. (Στις περισσοτερες γλωσσες υψηλου επιπεδου αυτο εκφραζεται ως t = s) Η επικεφελιδα της συνατησης εχει την μορφη:

```
    void xstrcpy(char t[],char s[])
```

Λυση 1

```
void xstrcpy(char t[],char s[])
{
    int i = 0;
    while (s[i] != '\0') {
        t[i] = s[i];
        i++;
    }
    t[i] = '\0';  /* Τερματισμος του string */
}
```

Με χρηση δεικτων 1:

```
void xstrcpy(char *sr, char *s)
{
    while(*s != '\0') {
        *sr = *s;
        sr++;
```

```
        s++;
    }
    *sr = '\0';
}
```

Με χρηση δεικτων 2:

```
void xstrcpy(char *sr, char *s)
{
    while(*s)
        *sr++ = *s++;
    *sr = '\0';
}
```

Με χρηση δεικτων 3:

```
void xstrcpy(char *sr, char *s)
{
    while(*sr++ = *s++);
}
```

-- Να γραφει προγραμμα το οποιο διαβαζει απο το τερματικο strings (αντιπροσωπευουν ονοματα μαθητων μιας ταξης) με την cin και αποθηκευει τα strings σε εναν πινακα χαρακτηρων δυο διαστασεων (καθε γραμμη του πινακα κρατα ενα ονομα). Στην συνεχεια να γραφει μια συναρτηση η οποια εμφανιζει τον διδιαστατο πινακα. Η επικεφαλιδα της συναρτησης εχει την εξης εμφανιση:

void prnstr2D(char sa[][41],int n)

Μεγιστος αριθμος στηλων στον πινακα 41.

Οπου n = Πληθος γραμμων του πινακα.

Λυση

```
#include <iostream.h>
#include <string.h>  // Για να χρησιμοποιησουμε την strlen, και strcpy

void prnstr2D(char sa[][41],int n);   // Πρωτοτυπο

int main()
{
    char buffer[512];
    char names[12][41];
    i = 0;
    while (1) {
        cin >> buffer;
        if (strlen(buffer) > 40) {
            cout << "Δωσε ονομα μεχρι 40 γραμματα\n";
            continue;
        }
        strcpy(names[i++],buffer);
        if (i > 11)
            break;
    }
    prnstr2D(names,12);
}

void prnstr2D(char sa[][41],int n)
{
```

```
    for(i = 0; i < n; i++)
        cout << sa[i] << endl;
}
```

-- Να γραφει μια συναρτηση η οποια αντιγραφει τους n πρωτους χαρακτηρες ενος string s στο string t. Η επικεφελιδα της συνατησης εχει την μορφη:

```
    void xstrncpy(char t[],char s[],int n)
```

Λυση 1

```
void xstrncpy(char t[],char s[],int n)
{
    int i = 0;
    while (s[i] != '\0' && n <= 1) {
        t[i] = s[i];
        i++;
        n--;
    }
    t[i] = '\0';
}
```

Λυση 2

Με χρηση δεικτων.

```
void xstrncpy(char *t,char *s,int n)
{
    while ( n-- > 0 && (*s) )
        *t++ = *s++;

    *t = 0;  // ιδιο με το *t = '\0';
}
```

-- Να γραφει η συναρτηση len η οποια επιστρεφει το μηκος ενος string. Στο μηκος δεν υπολογιζεται ο τερματικος χαρακτηρας. Η επικεφελιδα της συνατησης εχει την μορφη:

```
    int len(char s[])
```

Λυση 1

```
int len(char s[])
{
    int i = 0;
    while (s[i] != '\0') i++;
    return(i);
}
```

Λυση 2: Με δεικτες

```
int len(char *s)
{
    int i = 0;
    while ( *(s+i) != '\0') i++;
    return(i);
}
```

Λυση 3: Με δεικτες

```
int len(char s[])
{
    int i = 0;
```

```
    while ( *s++ != '\0') i++;
    return(i);
}
```

-- Να γραφει μια συναρτηση xstrcat η οποια προσθετει σε ενα string st2 στο string st1 (Στις περισσοτερες γλωσσες υψηλου επιπεδου αυτο εκφραζεται ως st1 = st1 + st2) Η επικεφελιδα της συνατησης εχει την μορφη:

```
    void xstrcat(char st1[], char st2[])
```

Λυση

```
include <iostream.h>
#include <string.h>
void xstrcat(char st1[], char st2[]);
int main()
{
    char a[100],b[30];

    a[0] = b[0] = '\0';   // Μηδενισμος των strings
    strcpy(a,"Adam");
    strcpy(b," Eva");

    xstrcat(a,b);

    cout << "Ενωση δυο string= |" << a << "|" << endl;
    return(0);
}

void xstrcat(char st1[], char st2[])
{
    int i=0, j=0;

    while(st1[i++] != '\0');   // Ο δεικτης οδηγειται στο
                               // τελος του string.
    while(st2[j] != '\n')      // Προσθεσε το st2 στο st1
        st1[i++] = st2[j++];   // χαρακτηρα χαρακτηρα.
    st1[i] = '\0';             // Τερματισμος του string.
}
```

Αποτελεσμα

```
|Adam Eva|
```

-- Να γραφει μια συναρτηση at η οποια θα επιστρεφει εναν ακεραιο που δειχνει την θεση που βρισκεται ενα string s1 μεσα στο s2. Αν πχ. s1 = "ABC" s2 = "aAcABCD" η συναρτηση επιστρεφει 4 (ο πρωτος χαρακτηρας του "ABC" στο s2 ειναι στο κελι με δεικτη 3).

```
int at(char s1[], char s2[])
{
    int i, j, x, ls1, ls2;
    if( s1[0] == '\0' || s2[0] == '\0')
        // return = 0 εαν ενα απο τα δυο string ειναι κενο
        return(0);

    ls1 = strlen(s1);
    ls2 = strlen(s2);

    i = 0;
```

```
    while( i + ls1 <= ls2 )  {
       x = i;
       // Οσο οι χαρακτηρες ειναι ιδιοι συνεχισε την συγκριση
       for(j = 0; s1[j] == s2[i] && s1[j] != '\0'; i++, j++);

       if( j >= ls1 )    // Ο δεικτης του s1 εφθασε στο τελος
          return( x+1 ); // Επεστρεψε τον δεικτη του s2 + 1
       i = x + 1;
    }
    return(0);  // Δεν βρεθηκε το s1 στο s2
}
```

-- Να γραφει μια συναρτηση rat η οποια θα επιστρεφει εναν ακεραιο που δειχνει την θεση που βρισκεται ενα string s1 μεσα στο s2 αλλα απο τα δεξια. Αν πχ. s1 = "ABC" s2 = "aABCcAqABCD" η συναρτηση επιστρεφει 8 (ο πρωτος χαρακτηρας του "ABC" στο s2 ειναι στο κελι με δεικτη 7).

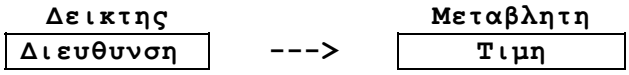
-- Να γραφει μια συναρτηση zero_strarr η οποια μηδενιζει εναν πινακα string με n γραμμες και 41 στηλες.

```
void zero_strarr(char s[][41],int n);

int main()
{
    char ka[5][41];

    zero_strarr(ka,5)

    return(0);
}

void zero_strarr(char s[][41],int n)// Υποχρεωτικα δηλωνονται
{                                   // οι στηλες του πινακα.
    int i;
    for(i = 0; i <= n; i++)  // Το πρωτο κελι καθε γραμμης
        s[i][0] = '\0';      // παιρνει την τιμη '\0'
}
```

Δεικτες (pointers)

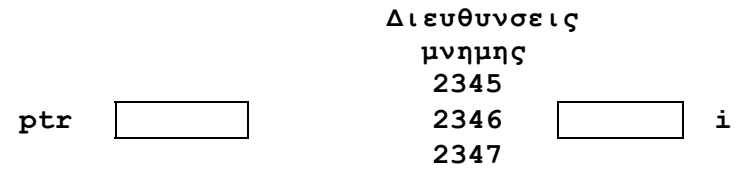
Δεικτης (pointer) ειναι ενας τυπος δεδομενων ο οποιος κρατα μια διευθυνση μιας μεταβλητης στην μνημη (οχι την τιμη της μεταβλητης). Ενας δεικτης ειναι τυπος unsigned. Οι δεικτες ειναι πολυ ευχρηστοι για την επεξεργασια των strings.
Ενας δεικτης οριζεται ως εξης:

```
τυπος *ονομα_δεικτη;
```

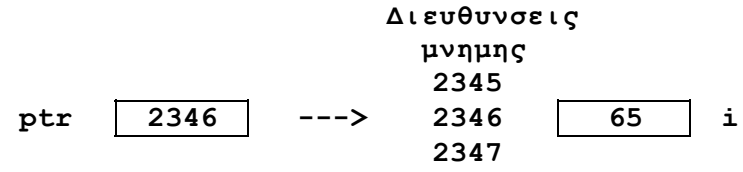


α) `char *str;` Η μεταβλητη str ειναι δεικτης σε ενα κελι μνημης που μπορει να κρατησει εναν χαρακτηρα.

```
β) int i;      // Ακεραιος i.
   int *ptr;   // Ο ptr ειναι δεικτης σε εναν ακεραιο.
```




```
i = 65;   // Ο i παιρνει την τιμη 65
ptr = &i; // Ο ptr παιρνει την διευθυνση του i
```



Για να εχουμε προσβαση στο περιεχομενο της διευθυνσης που δειχνει ενας δεικτης χρησιμοποιουμε το (*).

Αφ' οσον ο δεικτης ptr δειχνει στο i μπορουμε να:

- Ανακτησουμε την τιμη του i

```
cout << i ;     // Αμεσα, θα εμφανισθει 65
cout << *ptr;   // Εμεσως, θα εμφανισθει 65
```

- Απονομουμε τιμη στο i

```
i = 69;      // Αμεσα
*ptr = 102; // Εμεσως
```

Παραδειγμα 1

```
#include <iostream.h>
int main()
{
    int i,j;
    int *ptr;  // Δηλωση δεικτη σε ακεραιο
    i = 65;
    ptr = &i;  // Ο δεικτης δειχνει στο i
    j = *ptr;  // Ο j = με τον χαρακτηρα που δειχνει ο ptr
```

```
    cout << "i=" << i << endl;
    cout << "j=" << j << endl;
    return(0);
}
```
Αποτελεσμα
```
i=65
j=65
```

Παραδειγμα 2
```
#include <iostream.h>
int main()
{
   int i; int *ptr;

   ptr = &i; // Ο δεκτης ptr δειχνει στον ακεραιο i.
   *ptr = 69;// Το κελι που δειχνει ο ptr παιρνει την τιμη 69
             // Το κελι αυτο ομως εχει "ονομα" i (αρα ο
             // i = 69.
   cout << "i=" << i << endl;
   return(0);
}
```
Αποτελεσμα
```
i=69
```

Παραδειγμα 3
```
#include <iostream.h>
int main()
{
   int a;
   int *ptr = &a;// Δωσε τιμη στον δεικτη την διευθυνση του a

   a = 65;
   if (a == *ptr)  // η σχεση ειναι αληθης
       cout << "Οι τιμες ειναι ισες\n";
   return(0);
}
```
Αποτελεσμα
```
Οι τιμες ειναι ισες
```

Ολοι οι δεικτες χρησιμοποιουν δυο τιμες οταν χειριζονται μεταβλητες και το περιεχομενο τους. Η πρωτη ειναι η θεση (διευθυνση στην μνημη) που δειχνει ο δεικτης. Η δευτερη ειναι η τιμη που περιεχεται στην διευθυνση που δειχνει ο δεκτης.

```
#include <iostream.h>
int main()
{
    int  i;  // Ακεραιος
    int *ptr;  // Δεικτης σε ακεραιο

    ptr = &i;

    for(i = 0; i <= 10; i++) {
        cout << "Τιμη του i = " << i <<
                "Διευθυνση του i = " << &i << endl;
```

```
        cout << "Διευθ. που δειχνει ο ptr = " << ptr <<
                "και το περιεχομενο αυτης της διευθ. = " <<
                 *ptr << endl;
    }
    return(0);
}
```

--

```
#include <iostream.h>
int main()
{
    char ar[10], *ptr;

    strcpy(ar,"Lotus");
    ptr = ar;   // Το ptr δειχνει στο πρωτο χαρακτηρα του ar.
    while (*ptr != '\0') {
        cout << *ptr;
        ptr++;    // Προχωρα στον επομενο χαρακτηρα.
    }
    return(0);
}
```

```
        ar
  0    | L  | <- ptr + 0
  1    | o  | <- ptr + 1
  2    | t  |      .
  3    | u  |      .
  4    | s  |      .
  5    | \0 | <- ptr + 5
```

Το `ptr = ar;` ειναι ισοδυναμο με: `ptr = &ar[0];`

Πινακες απο Δεικτες

Οι πινακες απο δεικτες (arrays of pointers) οριζονται ως εξης:

τυπος * ονομα_πινακα[αριθμος_κελλιων]

a) `char *names[10];`
Πινακας απο 10 κελια. Σε καθε κελι μπορει να αποθηκευθει ενας δεικτης που δειχνει σε χαρακτηρα.

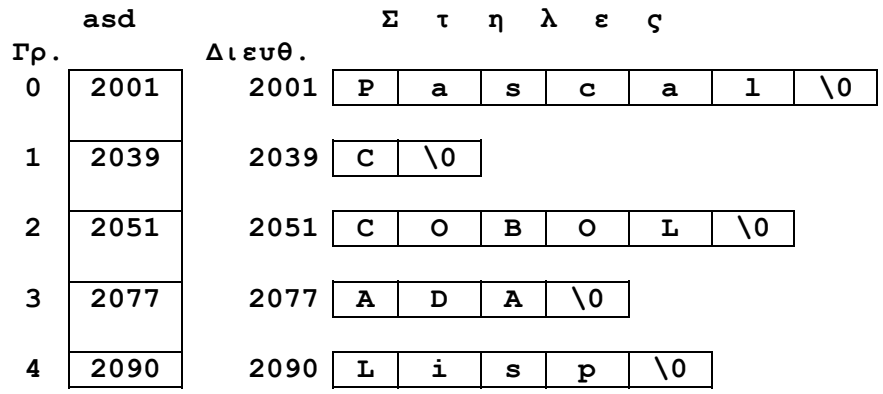
b) `char *cities[] = { "Thessaloniki", "Patra", "Katerini" };`
Πινακας απο δεικτες σε χαρακτηρα με τρια (το μεγεθος το βρισκει ο μεταφραστης) κελια. Ο πρωτος δεικτης δειχνει στο χαρακτηρα T του string Thessaloniki, ο δευτερος στο P του string Patra κ.ο.κ.

Δηλωση πινακα δεικτων σε χαρακτηρες asd πεντε (5) θεσεων (γραμμων):

```
char *asd[] = {
       "Pascal",
       "C",
       "COBOL",
       "ADA",
       "Lisp"
};
```

- Καθε δεικτης δειχνει σε ενα string.

- Οι δεικτες ειναι σε διαδοχικες θεσεις στην μνημη.
- Τα strings δεν ειναι σε διαδοχικες θεσεις μνημης.



Περασμα **πινακα δεικτων ως παραμετρο σε συναρτηση** γινεται οπως γινεται σε ολους τους πινακες

Α) Με αγκυλες **[]**

```
void foo(char* []);  // Πρωτοτυπο
:
char* pap[2] = { "Ναι", "Οχι" };  // Πινακας δεικτων
:
foo(pap);            // Κληση
:
void foo(char* pd[]) // Επικεφαλιδα
{
   //
}
```

Β) Με δεικτη *****

```
void boo(char* *);     // Πρωτοτυπο
:
char* pap[2] = { "Ναι", "Οχι" };
:
foo(pap);              // Κληση
:
void boo(char* *pd)    // Επικεφαλιδα
{
   //
}
```

(Εδω εμφανιζεται ενας νεος τυπος ο ****** που ειναι δεικτης σε δεικτες.)

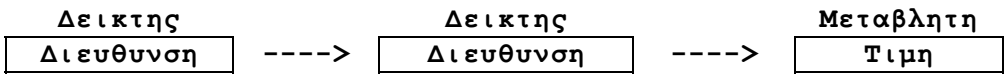
```
#include <iostream.h>
void disp_2D(char *[],int);
int main()
{
    char *ptr; int i;
    char *asd[] = {    // Δηλωση του πινακα των δεικτων
           "Pascal",   // στα 5 strings
           "C",
           "COBOL",
           "ADA",
           "Lisp"
    };
```

```
    printf("--1--\n");
    cout << asd[3] << endl;    // θα εμφανισθει ADA

    printf("--2--\n");
    disp_2Ds(asd, 5);          // εμφανιση ολου του πινακα

    printf("--3--\n");
    ptr = &asd[0][0];
    cout << ptr << endl;       // θα εμφανισθει Pascal

    printf("--4--\n");
    ptr = asd[4];
    cout << ptr << endl;       // θα εμφανισθει Lisp

    printf("--5--\n");
    ptr = asd[3];
    cout << *(ptr+1) << endl; // θα εμφανισθει D απο το ADA
    return(0);
}

void disp_2Ds(char *a[],int n)
{
   int i = 0;
   while( i < n )
      cout << asd[i++] << endl;
}
```

Δεικτες σε δεικτες

Δεικτης σε δεικτη (pointer to pointer) ειναι ενας τυπος δεδομενων ο οποιος κρατα μια διευθυνση ενος δεικτη. Ενας δεικτης ειναι τυπος unsigned. Οι δεικτες σε δεικτες ειναι πολυ ευχρηστοι για την επεξεργασια πινακων απο strings.
Ενας δεικτης οριζεται ως εξης:

```
    τυπος **ονομα_δεικτη;
```



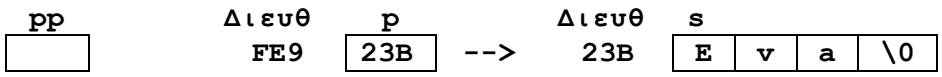
α) `char **str;` Η μεταβλητη str ειναι δεικτης σε ενα δεικτη ο οποιος δειχνει σε ενα κελι μνημης που μπορει να κρατησει εναν χαρακτηρα.

β) `int **ptr;` Ο ptr ειναι δεικτης σε δεικτη που δειχνει σε εναν ακεραιο.

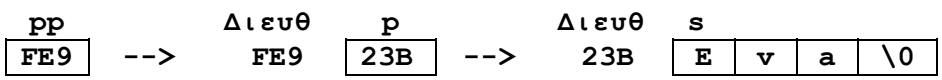
Παραδειγμα

```
char s[4] = { "Eva"};  // Πινακας s
char *p;               // Δεικτης p
char **pp;             // Δεικτης σε δεικτη pp
```


```
p = s;   // Ο δεικτης p δειχνει στον πινακα s
```




```
pp = &p;  // Ο δεικτης pp δειχνει στον δεικτη p
```




```
cout << *pp << endl; // Ανακτηση τιμης του s
                     // Εμφανιζεται το string Eva
---
```

```
#include <iostream.h>
int main()
{
    char s[10] = {"Eva"};
    char    *p;
    char  **pp;

    p = s;
    cout << " Δ: " << p << " " << *p << endl;

    pp = &p;
    cout << "ΔΔ: " << *pp << " " << **pp << endl;

    return(0);
}
```

Αποτελεσμα

```
 Δ: Eva E
ΔΔ: Eva E
```

Δεικτες σε συναρτησεις (pointers to functions)

Αν και οι συναρτησεις δεν ειναι μεταβλητες καταλαμβανουν ενα χωρο στην μνημη. Η διευθυνση που βρισκεται η συναρτηση ειναι το "σημειο εισοδου" (σημειο αρχης εκτελεσης) της συναρτησης. Η διευθυνση μιας συναρτησης ειναι το ονομα της συναρτησης (χωρις παρενθεσεις) και μπορει να δοθει σε εναν δεικτη. Κανοντας χρηση αυτου του δεικτη μπορουμε να εκτελεσουμε την συναρτηση.

α) `void (*pf)();`
Ο δεικτης pf ειναι μπορει να δειξει σε void συναρτηση χωρις παραμετρους

β) `int (*ptr)(int);`
Ο ptr μπορει να δειξει σε int συναρτηση με μια int παραμετρο.

γ) `char* (*pF)(char *,char,int);`
Ο pF μπορει να δειξει σε συναρτηση που καλειται με τρεις πραμετρους και επιστρεφει εναν char *

δ) `void (*p[2])();`

Πινακας (με δυο κελλια) απο δεικτες σε void συναρτησεις

-- Παραδειγμα δεικτη σε συναρτηση.

```
#include <iostream.h>
#include <string.h>

int main()
{
   long unsigned (*pfn)(char *);// Δηλωση δεικτη σε συναρτηση
   long unsigned a;

   pfn = strlen;   // Ο δεικτης δειχνει στην συναρτηση strlen

   a = (*pfn)("Rainbow"); // Κληση της συναρτησης (εμεσως)
                          // Ευρεση μηκους στρινγ

   cout << "Μηκος στρινγ " << a << endl;

   return(0);
}
// g29.cpp //
```

Αποτελεσμα

```
Μηκος στρινγ 7
```

-- Παραδειγμα δεικτη σε συναρτηση

```
#include <iostream.h>

void disp();       // πρωτοτυπο συναρτησης

int main()
{
    void (*pf)();// Δηλωση δεικτη σε void συναρτηση

    pf = disp;   // Ο δεικτης δειχνει την συναρτηση disp

    pf();        // Κληση της συναρτησης disp μεσω του δεικτη

    return(0);
}

void disp()
{
    cout << "Good news" << endl;
}
// g25.cpp //
```

Αποτελεσμα

```
Good news
```

-- Παραδειγμα, Πινακα απο δεικτες σε συναρτησεις.

```
#include <iostream.h>

void disp();
void out();
```

```
int main()
{
    void (*pf[2])();  // Δηλωση πινακα απο δεικτες σε void
                      // συναρτησεις
    pf[0] = disp; // Ο δεικτης[0] δειχνει την συναρτηση disp
    pf[1] = out;  // Ο δεικτης[1] δειχνει την συναρτηση out

    pf[0]();     // κληση της συναρτησης disp μεσω του δεικτη
    pf[1]();     // κληση της συναρτησης out  μεσω του δεικτη

    return(0);
}

void disp()
{
    cout << "Good news" << endl;
}

void out()
{
    cout << "Bad news" << endl;
}
// g26.cpp //
```

Αποτελεσμα

```
Good news
Bad news
```

-- Να γραψετε μια συναρτηση με την εξης επικεφαλιδα

```
int myget(char *m, char *s, int (*fn) (char *))
```

η οποια θα χρησιμοποιηθει για την εισαγωγη και ελεγχο στοιχειων απο το πληκτρολογιο.

α) Η παραμετρος m ειναι το στρινγ προτροπης (πχ. Δωσε εναν αριθμο ή Δωσε το τηλεφωνο).

β) Ολα τα στοιχεια εισαγονται ως στρινγ στην δευτερη παραμετρο s.

γ) Η τριτη παραμετρος ειναι ενας δεικτης σε **int** συναρτηση με μια παραμετρο στρινγ. Ο δεικτης πρεπει να δειχνει στην καταληλη συναρτηση οπου θα γινεται ελεγχος εγκυροτητος για το στοιχειο που εισαγεται.

Κανοντας χρηση της συναρτησης myget να εισαγεται τα παρακατω στοιχεια ενος σπουδαστη για το μαθημα Προγραμματισμος Η/Υ.

1) ονοματεπωνυμο (δεν υπαρχει ελεγχος εγκυροτητας, μια συναρτηση (fno) που ελεγχει το ονομα επιστρφει παντα EGKYRO)

2) βαθμος εργαστηριου (εγκυρος βαθμος απο 3 εως 6, η εγκυροτητα ελεγχεται στην συναρτηση fne),

3) βαθμος θεωριας (εγκυρος βαθμος απο 2 εως 4, η εγκυροτητα ελεγχεται στην συναρτηση fnt)

```
#include <iostream.h>
#include <iomanip.h>          // Χρειαζεται για το setw
#include <stdlib.h>           //   ..  ..   ..  .. atoi

#define EGKYRO  1
#define AKYRO   0

int myget( char *m, char *s, int (*fn)(char *) );
```

```
int fno(char *s);
int fne(char *s);
int fnt(char *s);

int main()
{
    char name[20];
    char bae[6];
    char bat[6];

    myget("Ονοματεπωνυμο...........: ",name,fno);
    myget("Βαθμος Εργαστηριου (3-6): ",bae, fne);
    myget("  >>   Θεωριας.... (2-4): ",bat, fnt);

    // Υπολογισμος Μ.Ο βαθμολογιας
    float b = (atoi(bae)+atoi(bat))/2.0;

    cout << "\n-- Τελικη Βαθμολογια --\n";
    cout << "Ονοματεπωνυμο: " << name << endl;
    cout << "Βαθμος.......: " << setw(5) << b << endl;

    return(0);
}

int myget( char *m, char *s, int (*fn)(char *) )
{
    while (1) {
        cout << m;
        cin >> s;

        if (fn(s))  // Κληση συναρτησης ελεγχου με δεικτη
                    // στην συναρτηση
           break;
    }
}

int fne(char *s)  // Ελεγχος βαθμου εργαστηριου
{
    int g;
    g = atoi(s);
    if (g < 3 || g > 6) {
        cout << "     Λαθος. Βαμθος Εραγαστηριου (3-6)\n";
        return(AKYRO);
    }
    return(EGKYRO);
}

int fnt(char *s)  // Ελεγχος βαθμου θεωριας
{
    int g;
    g = atoi(s);
    if (g < 2 || g > 4) {
        cout << "     Λαθος. Βαθμος θεωριας (2-4)\n";
        return(AKYRO);
    }
```

```
    return(EGKYRO);
}

int fno(char *s)  // Ελεγχος ονοματος
{
    return(EGKYRO);
}
// g27.cpp //
```

-- Στο προηγουμενο προγραμμα να προσθεσετε ενα ή και περισσοτετα απο τα παρακατω:
1) Ενα ακομη στοιχειο για εισαγωγη (ως στρινγ) το αριθμο των παρουσιων του σπουδαστη στο εργαστηριο (εγκυρος αριθμος απο 8 εως 13) Ο ελεγχος θα γινεται σε μια συναρτηση fnp
2) Ενα ακομη στοιχειο για εισαγωγη (ως στρινγ) το φυλλο
του σπουδαστη (εγκυρος χαρακτηρας Α ή Κ) Ο ελεγχος θα γινεται σε μια συναρτηση fns
3) Η συναρτηση fno που ελεγχει το ονοματεπωνυμο να βελτιωθει ωστε να μη γινεται δεκτο ονοματεπωνυμο που περιεχει αριθμητικο χαρακτηρα, ειδικους χαρακτηρες, σημεια στηξης κλπ.
4) Προσθεσετε στην myget δυο παραμετρους x και y (συντεταγμενες της οθονης) ωστε η εισαγωγη ενος στοιχειου να γινεται σε σταθερο σημειο της οθονης.
5) Τα μηνυματα λαθους εισαγωγης να εμφανιζεται παντα στην γραμμη 24 της οθονης.
6) Ενα ακομη στοιχειο για εισαγωγη (ως στρινγ) η ημερομηνια ανακοινωσης της τελικης βαθμολογιας στον σπουδαστη (πληρης ελεγχος ημερων, μηνων, δισεκτων ετων κλπ) Ο ελεγχος θα γινεται σε μια συναρτηση fnh

void και void δεικτης

Ο **void** ειναι ενας τυπος δεδομενων που δηλωνει την απουσια τυπου δεδομενων. Οταν μια συναρτηση δεν εχει παραμετρους τοτε χρησιμοποιειται το void για να δειξει οτι λειπουν παραμετροι. Επισης οταν μια συναρτηση δεν επιστρεφει καμμια τιμη τοτε η συναρτηση ειναι τυπου void.

α) `char f(void);` Η συναρτηση f δεν εχει καμμια παραμετρο
ιδιο με:
`char f();`

β) `void g(int x);` Η συναρτηση g δεν επιστρεφει καμμια τιμη

γ) `void f(void);` Η συναρτηση f δεν εχει παραμετρους δεν επιστρεφει τιμη

Δεν μπορει να δηλωθει μεταβλητη τυπου void:
`// ΛΑΘΟΣ  void a;`

Μπορουμε να δηλωσουμε εναν **δεικτη τυπου** **void**, ο οποιος ειναι δεικτης σε ακαθοριστο τυπο δεδομενων
α) `void* vPtr;` Ο δεικτης vPtr μπορει να δειξει σε 'καποιο' τυπο.

β) `void* f(void*);` Η συναρτηση f εχει ως παραμετρο εναν δεικτη void και επιστρεφει εναν δεικτη void

-- Παραδειγμα

```
#include <iostream.h>

void* func(void);     // Δεν εχει καμμια παραμετρο και
                      // επιστρεφει void*
int main()
{
    int  *p;
    char *pch;

    p = (int *) func(); // Μετατροπη απο void* σε int*
    cout << "*p = " << *p << endl;

    pch = (char *) func(); // Μετατροπη απο void* σε char*
    cout << "*pch = " << *pch << endl;

    return(0);
}

void* func(void)
{
    int x = 65;
    int *px = &x;
    return( (void*) px); // Μετατροπη απο int* σε void*
}
// g92.cpp //
```

Τελεστες new και delete

Ο τελεστης **new** καταλαμβανει δυναμικα (κατα την εκτελεση) μνημη απο το heap (χωρος μνημης που διατιθεται για προσωρινη χρηση κατα την διαρκεια εκτελεσης ενος προγραμματος)

Ο **new** επιστρεφει εναν δεικτη στη διευθυνση της καταλειφθεισης μνημης. Ο δεικτης δειχνει στο ακτικειμενο ή στα αντικειμενα. Εαν δεν καταστει δυνατον να καταλαβει τον οριζομενο χωρο στην μνημη τοτε επιστρεφει την τιμη NULL.

Συνταξη:

α) ***δεικτης*** **= new** ***αντικειμενο***

Ο *δεικτης* δειχνει σε ενα απλο αντικειμενο (int, char κλπ) ή συνθετο (struct, class)

β) ***δεικτης*** **= new** ***αντικειμενο*** **[*num*]**

Ο *δεικτης* δειχνει σε ενα πινακα απο αντικειμενα με *num* στοιχεια.

Ο τελεστης **delete** ελευθερωνει την μνημη που ειχε καταλειφθει με τον τελεστη new.

```
α) delete δεικτης;
β) delete [] δεικτης;  // για πινακες
```

Παραδειγματα:

a) Δημιουργια και καταστροφη ενος ακεραιου δυναμικα

```
int *p;          Δεικτης.
:
p = new int;     Ο δεικτης δειχνει στον ακεραιο.
:
delete p;        Ελευθερωση της μνημης.
```

b) Δημιουργια ενος πινακα ακεραιων με 10 στοιχεια.

```
int *ptr;
:
p = new int [10];     Ο δεικτης δειχνει στον πρωτο απο τα 10 στοιχεια
:                     ακεραιων
// Επεξεργασια
:
delete [] p;          Ελευθερωση της μνημης
```

c) Πινακας δυο διαστασεων.

```
double (*p) [40] = new double [5][40];
:
delete [40] p;
// 2ος τροπος:
double (*p) [40];
p = new double [5][40];
```

d) Πινακας χαρακτηρων

```
char *s;
:
s = new char [256];
:
delete [] s;
```

e) Αρχικη τιμη σε αντικειμενο

```
int *p = new int (10);          Ο *p παιρνει την τιμη 10
char *ch = new char ('a');      Ο *ch παιρνει την τιμη 'a'
```

f) Δεικτης σε συναρτηση του επιστρεφει ακεραιο.

```
int (*p) () = new (int ());
```

γ) Πινακας απο 7 δεικτες σε συναρτησεις που επιστρεφουν ακεραιο.

```
int (**p) () = new (int (*[7]) ());
```

-- Δημιουργια πινακα ακεραιων 10 στοιχειων δυναμικα. Εμφανιση του πινακα στην οθονη. Διαγραφη του πινακα.

```
#include <iostream.h>
int main()
{
   int *p;

   p = new int [10];   // Δημιουργια του πινακα
   if ( p == NULL ) {
      cout << "Αδυνατη η δημιουργια δυναμικου πινακα\n";
      return(1);
   }

   // Εισαγωγη αριθμων στον πινακα
   for( int i = 0; i < 10; i++)
      cin >> *(p+i);

   // Εμφανιση του πινακα
   for(int i = 0; i < 10; i++)
      cout << *p++ << " ";

   delete [] p;   // Διαγραφη του πινακα

   return(0);
}
```

-- Δημιουργια πινακα απο 5 στρινγς δυναμικα (πινακας χαρακτηρων δυο διαστασεων). Εισαγωγη των στρινγς απο το πληκτρολογιο. Εμφανιση του πινακα των στρινγς με την συναρτηση disp (Προσεξετε πως περνα ως παραμετρο ο δεικτης p) Εμφανιση του πινακα στο κυριως προγραμμα. Τελος διαγραφη του πινακα.

```
#include <iostream.h>
void disp(char (*p)[100], int n); // Πρωτοτυπο
int main()
{
   char (*p)[100]; // Δηλωση δεικτη σε πινακα δυο διαστασεων

   p = new char [5][100];   // Δημανικη δημιουργια
   if ( p == NULL ) {
      cout << "Adynath dhmioyrgia dynamikoy pinana"
              " χαρακτηρων δυο διαστασεων\n";
      return(1);
   }
```

```
   // Εισαγωγη 5 στρινγ
   for( int i = 0; i < 5; i++)
      cin >> *(p+i);

   disp(p,5);  // Εμφανιση των στρινγ με συναρτηση

   // Εμφανιση των στρινγ
   for(int i = 0; i < 5; i++)
      cout << *p++ << " ";

   delete [] p;   // Διαγραφη του πινανα
   return(0);
}

void disp(char (*p)[100],int n)
{
   for( int i = 0; i < 5; i++)
      cout << p[i] << " ";  // p[i] ιδιο με *(p+i)
   cout << "\n";
}
```

const σταθερες

Ορισμος σταθερων με τον προσδιοριστης **const**.

const *ονομα_σταθερας = τιμη;*

Η τιμη συνδεεται με το ονομα της σταθερας. Η τιμη αυτη δεν μπορει να αλλαξει μεσα στο προγραμμα.

Παραδειγματα:

```
α) const int mines = 12;
β) const char YesNo = 'N';
γ) const int Mar = 25, Oct = 28;
```

Στα παρακατω παραδειγματα το const συνδεεται με δυο αντικειμενα (ο δεικτης, και η στρινγ σταθερα) αναλογα με τον τροπο γραφης:

```
α) const char *ptr = "thema";
```

- Το στρινγ "thema" δεν μπορει να αλλαξει. Το const συνδεεται με το στρινγ.
- Ο ptr μπορει να αλλαξει.

```
β) char * const ptr = "thema";
```

- Ο ptr δεν μπορει να αλλαξει. Το const συνδεεται με τον δεικτη.
- Το στρινγ "thema" μπορει να αλλαξει.

```
γ) const char *const ptr = "thema";
```

-Ο ptr δεν μπορει να αλλαξει. Το ενα const συνδεεται με τον δεικτη.

-Το στρινγ "thema" δεν μπορει να αλλαξει. Το αλλο const συνδεεται με το στρινγ

Το const σε παραμετρο συναρτησης:

```
void copy(char *neo, const char *palio)
{
  // Το στρινγ palio δεν μπορει να αλλαξει στην συναρτηση.
  // Ετσι με τον προσδιοριστη const το στρινγ palio προστατευεται.
  ...
}
```

static μεταβλητες

Οι στατικες μεταβλητες δηλωνονται οπως οι "κοινες" μεταβλητες με το προθεμα **static.**

```
α) static int x;      // x = 0; εξ' ορισμου
β) static double d = 23.6;  // d = 23.6
```

Εαν δεν δωσουμε αρχικη τιμη σε μια στατικη μεταβλητη τοτε εξ' ορισμου (by default) παιρνει τιμη 0 (μηδεν).

Οι στατικες μεταβλητες ειναι προσπελασιμες μονο μεσα στο πηγαιο αρχειο ή στην συναρτηση που δηλωνονται.

Αν μια στατικη μεταβλητη δηλωθει μεσα σε μια συναρτηση τοτε διατηρει την τιμη της μεσα στην συναρτηση οσες φορες και αν εκτελεσθει η συναρτηση.

Αν μια στατικη μεταβλητη δηλωθει σε ενα πηγαιο αρχειο διατηρει την τιμη της αλλα δεν ειναι προσπελασιμη απο αλλο πηγαιο αρχειο.

-- Στο παρακατω προγραμμα η στατικη μεταβλητη x στην συναρτηση func παιρνει αρχικη τιμη 100 στην πρωτη κληση και προστηθεται ο αριθμος 13 (δηλ. x = 113). Στην δευτερη κληση το x ΔΕΝ παιρνει παλι την τιμη 100 (οπως συμβαινει με τις "κοινες" μεταβλητες) αλλά κρατα την τιμη 113 σε αυτην την τιμη προστηθεται το 13 και εχει ως αποτελεσμα να εμφανισθει x = 126 (δηλ. 113+13)

```
#include <iostream.h>

void func();

int main()
{
    func();
    func();
}

void func()
{
    static int x = 100;  // Στατικη μεταβλητη χ
    int y = 100;         // "Κοινη" μεταβλητη y
    x = x + 13;
    y = y + 13;
    cout << "Στατικη μεταβλητη χ =" << x << endl;
    cout << "Κοινη   μεταβλητη y =" << y << endl;
}
// g79.cpp //
```

Αποτελεσμα
```
Στατικη μεταβλητη χ =113
Κοινη   μεταβλητη y =113
Στατικη μεταβλητη χ =126       // Διατηρησε την τιμη της
Κοινη   μεταβλητη y =113
```

-- Στο επομενο προγραμμα η στατικη μεταβλητη x δηλωνεται global και ειναι γνωστη μονο μεσα στο πηγαιο αρχειο `g80.cpp`

```
// Πηγαιο αρχειο g80.cpp
#include <iostream.h>

static int x; // Στατικη μεταβλητη χ (ολικης εμβελειας-
              // global).  Ειναι προσπελασημη μονο σε αυτο το
              // πηγαιο αρχειο
void func();  // Πρωτοτυπο

int main()
{
   x = 400;  // Προσβαση στη χ
   func();
   func();
}

void func()
{
   x = x + 13;  // Προσβαση στη x
   cout << "Στατικη μεταβλητη x =" << x << endl;
}
```

Αποτελεσμα
```
Στατικη μεταβλητη x =413
Στατικη μεταβλητη x =426
```

register μεταβλητες

Οι register μεταβλητες δηλωνονται οπως οι "κοινες" μεταβλητες με το προθεμα **register.**

α) register int i;
β) register char ch;

Το **register** ζητα απο τον μεταφραστη να κρατηση την τιμη μιας μεταβλητης σε ενα register του επεξεργαστη και οχι στην μνημη οπου ολες οι αλλες κοινες μεταβλητες αποθηκευονται. Οι λειτουργιες σε **register** μεταβλητες γινονται πιο γρηγορα απο οτι σε "κοινες" μεταβλητες.

Μπορουμε να δηλωσουμε οσες μεταβλητες θελουμε ως **register** δεν ειναι ομως σιγουρο οτι ο μεταφραστης θα βρει **register** ελευθερους.

Οι **register** μεταβλητες συνηθως χρησιμοποιουνται σε θηλειες για ταχυτερη επεξεργασια.

-- Παραδειγμα χρησης μεταβλητων **register**. Η συναρτηση υπολογιζει το παραγοντικο ενος αριθμου (5! = 1*2*3*4*5)

```
#include <iostream.h>

int parag(register int n);

int main()
{
    cout << "Παραγοντικο του 5 ειναι = " << parag(5) << endl;
}

int parag(register int n)  // μεταβλητη register n
{
    register int i, tmp;   // μεταβλητες register i και tmp
    tmp = 1;
    for(i = 2; i <= n ; i++)
          tmp = tmp*i;
    return(tmp);
}
// g81.cpp //
```

Αποτελεσμα

```
Παραγοντικο του 5 ειναι = 120
```

extern μεταβλητες

Ο προσδιοριστης **extern** δηλωνει οτι μια μεταβλητη μοιραζεται τον ιδιο χωρο μνημης με μια αλλη μεταβλητη με το ιδιο ονομα και ιδιο τυπο που εχει ομως ορισθει σε αλλο πηγαιο αρχειο.

Αρχειο A (gA.cpp)

```
-----------------
void fnA1();
void fnB1();
int x;
char ch;
double ar[100];
main()
{
    fnA1();
    fnB1();
}

void fnA1()
{
    x = 13;
}

// gxx -c gB.cpp
// gxx -o a.exe gA.cpp gB.o
```

Αποτελεσμα

```
 x = 13
```

Αρχειο B (gB.cpp)

```
-----------------
#include <iostream.h>

extern int x;
extern double ar[100];
extern char ch;

void fnB1()
{
   cout << "x=" << x << endl;
}

void fnB2()
{
    x = x + 44;
}
```

Ασκησεις

-- Να γραφει μια συναρητση xstrlen η οποια επιστρεφει το μηκος ενος string (Χρησημοποιειστε δεικτες).

```
#include <iostream.h>
int xstrlen(char *s);
int main()
{
    char s[80;

    cin >> s;

    cout << "Μηκος = " << xstrlen(s) << endl;

   return(0);
}

int xstrlen(char *s)
{
    int cnt;
    for(cnt = 0; *s != '\0'; cnt++)
        s++;
    return(cnt);
}
```

Λυση 2

```
int xstrlen(char *s)
{
    int cnt = 0;
    while (*s++)
        cnt++ ;
    return(cnt);
}
```

-- Να γραφει μια συναρτηση zero_array οποια μηδενιζει ΟΛΑ τα στοιχεια ενος πινακα χαρακτηρων με n στοιχεια (κανετε χρηση δεικτων).

```
zero_array(int *ar,int n)
{
    while (--n >= 0)
        *ar++ = 0;
}
```

-- Να γραφει μια συναρτηση η οποια επιστρεφει το αθροισμα των n πρωτων στοιχειων ενος πινακα (κανετε χρηση δεικτων).

```
int sumar(int *p, int n)
{
    int s = 0;
    for(int i = 1; i <= n ; i++, p++)
         s = s + *p;
    return(sum);
}
```

-- Να γραφει μια συναρτηση η οποια επιστρεφει εναν δεικτη int ο οποιος δειχνει στον μεγιστο (max) εκ των n πρωτων στοιχειων ενος πινακα.

```
#include <iostream.h>
int *ptomax(int *, int);

int main()
{
    int ar[] = { 1, 3, 7, 2, 9, 4, 6, 5, 8 };
    int *p;

    // Ευρεση του max στα 7 πρωτα στοιχεια του πινακα ar.
    p = ptomax(ar,7);
    cout << "Ο max αριθμος ειναι ο" << *p << endl;
    return(0);
}

int *ptomax(int *x, int n)
{
    int *pm;
    pm = x;  // Θεωρουμε οτι στην πρωτη διευθυνση
             // βρισκεται ο max.
    for(int i = 2; i <= n; i++) {
        x++;
        if ( *x > *pm )
              pm = x;
    }
    return(pm);
}
```

-- Να γραφει μια συναρτηση η οποια εμφανιζει εναν πινακα ακεραιων δυο διαστασεων με g γραμμες και k στηλες. Στην συναρτηση γινεται χρηση δεικτων και οχι χρηση αγκυλων. (Επειδη γινεται χρηση δεικτων δεν ειναι αναγκαια αναγραφη του αριθμου στηλων στην επικεφαλιδα και στο πρωτοτυπο της συναρτησης οπως συμβαινει για πινακες δυο διαστασεων)

```
#include <iostream.h>
void emf(int *p, int g,int k);  // Πρωτοτυπο
int main()
{
    int a[2][3] = { 8, 4 , 6, 3, 7 ,5 };

    emf(&a[0][0], 2, 3);          // Κληση
    return(0);
}

void emf(int *p, int g, int k)   // Επικεφαλιδα
{
    int i,j;
    for(i = 0; i < g; i++) {
        for(j = 0; j < k; j++)
            cout << *(p + i*k + j) << " ";
        cout << endl;
    }
} // g98.cpp //
```

Αποτελεσμα
```
8 4 6
3 7 5
```

-- Να γραφει μια συναρτηση με την εξης επικεφαλιδα:
void proc(int *pa, int n, int *sum, float *mean, int *flag)
η οποια χειριζεται πινακες ακεραιων n στοιχειων (ο pa ειναι δεικτης στο πρωτο στοιχειο του πινακα). Η συναρτηση θα εχει ως "εξοδο" (κληση με αναφορα): α) το αθροισμα του πινακα, β) τον μεσο ορο, γ) μια "σημαια" με τιμη 1 εαν υπαρχει ο αριθμος 7 μεσα στον πινακα διαφορετικα η τιμη ειναι 0.

-- Να γραφει μια συναρτηση repl_char η οποια αντικαθιστα σε ενα string ολους τους χαρακτηρες που ειναι ισοι με το oldc με εναν αλλο newc.

```
#include <iostream.h>

void repl_char(char *s, char oldc, char newc);

int main()
{
    char str[100];

    cout << "Δωσε ενα string ";
    cin >> str;
    cout << "Πριν την αλλαγη " << str << endl;
    repl_char(str,'a','e');
    cout << "Μετα την αλλαγη " << str << endl;
    return(0);
}

void repl_char(char *s, char oldc, char newc)
{
    while (*s != '\0') {
        if (*s == oldc)
            *s = newc;
        s++;
    }
}
```

-- Να γραφει μια συναρτηση del_spaces η οποια αντιγραφει το περιεχομενο ενος string S σε ενα αλλο string T με την διαφορα οτι το T δεν περιεχει πλεοναζοντα κενα διατηματα μεταξυ των λεξεων που τυχον υπαρχουν στο S (δηλ. διαγραφει τα πλεοναζοντα κενα διαστηματα σε ενα string)
πχ. `"ενα    δυο    τρια"  -> "ενα δυο τρια"`

```
#include <iostream.h>
void del_spaces(char *,char *);
int main(void)
{
    char ch_arr[128], b[128];
    cout << "Δωσε ενα string: ";
    cin >> ch_arr;
    cout << "Το string που εδωσες ειναι το |" << ch_arr <<
            "|" << endl;
    del_spaces(b,ch_arr);
    cout << "Μετα την διαγραφη των κενων εγινε |" << b <<
```

```
            "|" << endl;
    return(0);
}

// Η συναρτηση αντιγραφει το s στο t και διαγραφει
// πλεοναζοντα κενα διαστηματα.
void del_spaces(char *t,char *s)
{
    while(*s) {
       if (*s == ' ') {    //  Το πρωτο "κενο" χρειαζεται
           *t++ = *s++;    //  για να χωριζουν οι λεξεις.
           // Προσπερασε τα περισευοντα "κενα".
           while (*s++ == ' ');
       }
       else
           *t++ = *s++;
    }
    *t = '\0';  // Τερματισμος του t.
}
```

Λυση πιο επεξηγηματικη

```
void del_spaces(char *t,char *s)
{
    while(*s != '\0') {
        if (*s == ' ')  {
            *t = *s;
            s = s + 1 ;
            t = t + 1 ;

            while (*s == ' ') s = s + 1;
        }
        else {
            *t = *s;
            s = s + 1 ;
            t = t + 1 ;
        }
    }
    *t = '\0';
}
```

-- Να γραφει μια συναρτηση empty η οποια επιστρεφει 1 εαν ενα string ειναι κενο (δεν περιεχει χαρακτηρες ή εχει μονο κενους χαρακτηρες πχ. "" ή " ") Διαφορετικα επιστρεφει 0.

```
#include <iostream.h>
int empty(char *);
int main(void)
{
    char b[128];
    char *s = "    ";  // s = με κενα διαστηματα.

    b[0] = '\0';       // Ο b μηδενιζεται

    if (empty(s) && empty(b))
        cout << "Και τα δυο string (s=|" << s << "|, "
```

```
                "b=|" << b << "|) ειναι κενα" << endl;

    strcpy(b,"  ABBA ");
    if(!empty(b))
        cout << "To string b=|" << b  <<
                "| τωρα ΔΕΝ ειναι κενο\n";
    return(0);
}

int empty(char *s)
{
    int i;
    while ( *s == ' ')  /* πέρνα τα διαστηματα */
       s++;
    i = (*s == '\0' ? 1 : 0 );
    return( i );
}
```

-- Να γραφει μια συναρτηση nullstr η οποια ελεγχει για το εαν ενα string ειναι null string (ο πρωτος χαρακτηρας ειναι '\0') Η συναρτηση ελεγχει για null string δηλ. "" Η συναρτηση επιστρεφει: οταν 1 = null string, και 0 = ΟΧΙ null string

```
int nullstr(char *s)
{
    return( *s == '\0' ? 1 : 0 );
}
```

-- Να γραφει μια συναρτηση revstr(sr,s) η οποια αναστρεφει την σειρα των χαρακτηρων του string s και τους τοποθετει στο string sr. Το αρχικο string s παραμενει ως εχει. **(πχ. "Corfu" -> "ufroC")**

// Χωρις Τεστ //

```
void revstr(char *sr,char *s)
{
    int ls;
    ls = strlen(s);
    s = s + ls - 1; // Τοποθετησε τον δεικτη στο τελος του s.
    while (ls-- > 0) {
        *sr = *s;
        sr++;
        s--;
    }
    *sr = 0;          // Τερματισμος του sr.
}
```

-- Να γραφει μια συναρτηση atc που επιστρεφει την θεση ενος χαρακτηρα μεσα σε ενα string.

```
#include <iostream.h>
int atc(char c,char *s);
int main(void)
{
    char arr[40], ch;
    int p;

    ch = 'i';
```

```
    strcpy(a,"This is the end");

    p = atc(ch,arr);

    cout << "Θεση του χαρακτηρα " << ch << "\n" <<
            "μεσα στο string |" << arr << "| ειναι " << p <<
            endl;

    return(0);
}

// Επιστρεφει την θεση του χαρ. c (1 char) μεσα στο στρινγ s)
int atc(char c,char *s)
{
    int i = 0;
    while ( *s != '\0' ) {
       i++;
       if (*s == c)
           return(i);
       s++;
    }
    return(0);
}
```

Αποτελεσμα

```
Θεση του χαρακτηρα i
μεσα στο string |This is the end| ειναι 3
```

-- Να γραφει μια συναρτηση ratc που επιστρεφει την θεση (απο δεξια) ενος χαρακτηρα μεσα σε ενα string. Η θεση μετραται απο τα δεξια προς τα αριστερα. πχ. στην ακολουθια ADAM αν ψαξουμε για τον χαρ. Α η συναρτηση πρεπει να επιστρεψει 2.

Λυση

```
int ratc(char c, char *s)
{
    int i = 1, ls;
    if (*s == '\0')
        return(0);

    ls = strlen(s);
    s = s + ls - 1;  // Τοποθετησε τον δεικτη στο τελος του s

    while ( i <= ls ) {
       if (*s == c)
           return( i );
       s--;
       i++;
    }
    return(0);
}
```

-- Να γραφει μια συναρτηση left(sr,s,n) η οποια αντιγραφει τους n πρωτους χαρακτηρες του s sto sr (απο τα αριστερα).

```
#include <iostream.h>
void left(char *sr, char *s, int n);
```

```
int main(void)
{
    char ar[100], b[100];
    strcpy(ar,"Run time library"); // Αρχικη τιμη στο string

    left(b, ar, 3); // Αντεγραψε τους 3 πρωτους χαρ. του ar
                    // στο b.
    cout << "String b=|" << b << "|" << endl;
    return(0);
}

// s = το αρχικο string απο το οποιο θα εξαχθει το sr.
void left(char *sr, char *s, int n)
{
    register int i;
    i = 0;
    if (n < 0)   // εαν ειναι αρνητικος -> θετικο
        n = -n;
    while (*s != '\0' && i <= n-1)  {
        *sr++ = *s++;
        i++;
    }
    *sr = 0;
}
```

-- Να γραφει μια συναρτηση right η οποια επιστρεφει ενα υπο-string (απο τα δεξια ενος string) n χαρακτηρων.

-- Να γραφει μια συναρτηση countc η οποια επιστρεφει την συχνοτητα με την οποια υπαρχει ενας χαρακτηρας ch μεσα σε ενα string.

```
#include <iostream.h>
int countc(char *, char);
int main(void)
{
    char sa[60]; int metr;

    strcpy(sa,"This is it");
    metr = countc(sa,'i');
    cout << "Συχνοτητα του i μεσα στο |" << sa <<
            "| ειναι " << metr << endl;
    return 0;
}

int countc(char *s, char c)
{
    int cnt = 0;
    while (*s) {
       if (*s == c)
           cnt++;
       s++;
    }
    return(cnt);
}
```

-- Να γραφει μια συναρτηση vall η οποια επιστρεφει την αριθμητικη τιμη (long) ενος string.

```
#include <stdio.h>

long vall(char *);

int main(void)
{
    long li;  char *s = "1234567";

    li = vall(s);
    cout << "To string |" << s << "| εχει τιμη " << li
         << endl;
    return(0);
}

long vall(char *tp)  /* li = vall(expS) */
{
    long retval;
    retval = 0;
    do {
        if (*tp >= '0' && *tp <= '9') {
            retval *= 10;
            retval += (*tp - '0');
        }
    } while (*tp++);

    return(retval);
}
```

-- Να γραφει αντιστοιχη συναρτηση vald η οποια υπολογιζει την αριθμητικη τιμη ενος string (πχ. "123.456") ως πραγματικο εχοντας ως οδηγο τον προηγουμενο αλγοριθμο.

-- Να γραφουν δυο συναρτησεις με τις οποιες ειναι δυνατον να προσθεσουμε ενα string ή εναν χαρακτηρα στο τελος ενος string. Να χρησιμοποιεισετε υπερφωρτωση συναρτησης.

-- Να γραφει μια συναρτηση ltrim η οποια επιστρεφει ενα string απηλλαγμενο απο τα κενα διαστηματα τα οποια βρισκονται στα αριστερα του string.
(πχ. `"     Hard disk  " -> "Hard disk  "`)

```
#include <iostream.h>
void ltrim(char *,char *);
int main()
{
    char a[128], b[128];

    strcpy(a,"      be yourself ");

    ltrim(b,a);
    cout << "To |" << a << "|εγινε |" << b << "|" << endl;

    return(0);
}

void ltrim(char *sr,char *s)
```

```
{
    while (*s == ' ')   /* απεφυγε τα κενα διαστηματα */
        s++;
    while ( *s != '\0' ) {
        *sr++ = *s++;
    }
    *sr = '\0';
}
```

Αποτελεσμα

```
To |      be yourself | εγινε |be yourself |
```

-- Να γραφει μια συναρτηση contain η οποια ελεγχει για το εαν καποιος χαρακτηρας του string f βρισκεται μεσα στο string s και τοτε επιστρεφει την θεση του χαρακτηρα μεσα στο s. Διαφορετικα επιστρεφει 0 (κανενας χαρακτηρας του f δεν υπαρχει μεσα στο s).

```
#include <iostream.h>
int contain(char *f,char *s);
int main(void)
{
    int pos; char a[128], b[128], *ptr;

    strcpy(a,"letter");
    strcpy(b,"assigned");

    ptr = &a[0];
    pos = contain(a, b);
    cout << "Ο πρωτος χαρακτηρας του |" << a
         << "| μεσα στο |" << b << "| ειναι " << pos << endl;

    ptr = ptr + pos - 1;
    cout << "Ο χαρακτηρας ειναι " << *ptr << endl;

    return(0);
}

// Return = θεση, if ενας char του f ειναι μεσα στο s
//                else return = 0
// πχ. (f="qwiert",s="Aapbis") -> return = 5 (θεση του i )
int contain(char *f,char *s)
{
    int b;
    if ( !(*f) || !(*s) )
        return(0); // Ενα απο τα δυο string ειναι null string
    // Ενας-ενας χαρατηρας του f ελεγχεται για το εαν υπαρχει στο s.
    while ( *f ) {
        if ( (b = atc(*f,s)) > 0 )
            // Η συναρτηση atc εχει αναλυθει παραπανω
            return(b);
        f++;
    }
    return(0);
}
```

-- Να γραφει μια συναρτηση strcpys η οποια αντιγραφει στο sr απο το το s n bytes. Εαν το μηκος του s ειναι μικροτερο του n τοτε να προσθετει κενα διαστηματα μεχρι το sr να εχει μηκος n.

```
#include <iostream.h>
void strcpys(char *sr, char *s, int n);
int main()
{
    char buf[128], tar[512];
    strcpy(buf,"So far so good");
    strcpys(tar,buf,20);

    cout << "String=|" << tar << "|  Μηκος=" << strlen(tar)
         << endl;
    strcpys(tar,buf,6);
    cout << "String=|" << tar << "|  Μηκος=" << strlen(tar)
         << endl;
    return(0);
}

// Αντεγραψε στο sr απο το το s n bytes;
// εαν μηκος του s < n τοτε προσθεσε κενα διαστηματα.
void strcpys(char *sr, char *s, int n)
{
    while ( (*s)  && n-- > 0  )  // Αντεγραψε n bytes
        *sr++ = *s++;
    while ( n-- > 0 )   // Εαν n > 0 τοτε αντεγραψε ' '
        *sr++ = ' ';
    *sr = 0;            // Τερματισμος του string
}
```

Αποτελεσμα

```
|So far so good      | μηκος=20
|So far| μηκος=6
```

-- Να γραφει ενα προγραμμα στο οποιο δημιουργει δυναμικα δυο πινακες ακεραιων Α και Β. Το πληθος των στοιχειων των πινακων διδεται απο το τερματικο. Στην σηνεχεια:

α) σε συναρτηση γινεται εισαγωγη ακεραιων απο το τερματικο στο πρωτο Α.

β) σε συναρτηση να αντιγραφει ο πινακας Α στον Β χωρις να περασουν τα στοιχεια που εχουν τιμη μηδεν.

γ) Να εμφανισθει το πληθος των στοιχειων του πινακα Β.

Λυση

```
#include <iostream.h>

void readA(int *, int);
int cpAtB(int *, int *, int);

int main()
{
    int num;
    cout << "Δωσε πληθος στοιχειων των πινακων Α και Β:";
    cin >> num;
    int *A = new int [num]; // Δυναμικη δημιουργια πινακα
    int *B;
    B = new int [num];
```

```
    readA(A,num); // Διαβασε απο το τερματικο num ακεραιους
    int r;
    r = cpAtB(B,A,num);    // Αντιγραφη του A στο B.
    cout << "Πληθος στοιχειων στο B =" << r << endl;

    delete [] B;
    delete [] A;

    return(0);
}

void readA(int *p, int n)
{
    for(int i = 0; i < n; i++, p++) {
        cout << "Δωσε τον " << i+1 << "ο αριθμο ";
        cin >> *p;
    }
}

int cpAtB(int *t, int *s, int n)
{
    int m = 0;
    for(int i = 0; i < n; i++)
        if (*s != 0) {
             *t++ = *s++;
             m++;
        }
        else s++;
    return(m);
}
// p49.cpp //
```

-- Να γραφη ενα προγραμμα το οποιο ελεγχει ολους τους χαρακτηρες απο 0x0 εως 0x7F, για το εαν οι χαρακτηρες ειναι isalnum, isaplha, isascii, iscntrl, isdigit, islower, ispunct, isspace, isprint, isupper, isxdigit και εμφανιζει για καθε χαρακτηρα τον τυπο σε συντομογραφια.

```
#include <iostream.h>
#include <ctype.h>
int main()
{
   int ch;
   for(ch = 0; ch < 0x7F; ch++) {
      cout << ch;
      cout << isprint(ch)  ? char(ch) : '\0';
      cout << isalnum(ch)  ? "AN" : "";
      cout << isalpha(ch)  ? "A"  : "";
      cout << isascii(ch)  ? "AS" : "";
      cout << iscntrl(ch)  ? "C"  : "";
      cout << isdigit(ch)  ? "D"  : "";
      cout << islower(ch)  ? "L"  : "";
      cout << ispunct(ch)  ? "PU" : "";
      cout << isspace(ch)  ? "S"  : "";
      cout << isprint(ch)  ? "PR" : "";
      cout << isupper(ch)  ? "U"  : "";
      cout << isxdigit(ch) ? "X"  : "";
```

```
      cout << endl;
   }
   return(0);
}
```

Αποτελεσμα

```
00            AS  C
01            AS  C
02            AS  C
...
40  @          AS            PU      PR
41  A  AN  A  AS                    PR  U  X
42  B  AN  A  AS                    PR  U  X
43  C  AN  A  AS                    PR  U  X
...
47  G  AN  A  AS                    PR  U
...
...
```

Παραλλαγη: Να αλλαξετε το προγραμμα ετσι ωστε το U (upper-case) και το L (low-case) να εμφανιζονται στην ιδια στηλη.

Εργασιες σε επιπεδο Bit

Το bit (binary digit) ειναι η μικροτερη μοναδα μνημης ενος Η/Υ (οι τιμες που μπορει να παρει ειναι 0 ή 1). Τα προγραμματα στην C/C++ συνηθως εχουν προσπελαση σε bits μεσω των ακεραιων τυπων char, short, int, long, και των δεικτων. Επισης η C/C++ διαθετει πεδια-bits (bit-fields) που μας επιτρεπει να εχουμε προσβαση σε ενα απλο bit. Ο πιο "μικρος" τυπος δεδομενων στην C/C++ ειναι ο char που εχει μεγεθος 1 byte (8 bits).

Χρηση

- Μπορουμε να αποθηκευσουμε δυαδικες (0 ή 1) μεταβλητες σε ενα μονο byte.
- Ορισμενες συσκευες αποστελουν πληροφοριες κωδικοποιημενες σε bits
- Ορισμενοι αλγοριθμοι κρυπτογραφησης εχουν προσβαση σε bits ενος byte.

Τελεστες

Οι τελεστες σε επιπεδο bit (bit-wise operators) επιδρουν (λειτουργουν) πανω στις μεταβλητες ή σταθερες κατα τετοιο τροπο σαν να ειναι πινακες bits.

Τελεστης	Περιγραφη	Παραδειγμα	
~	Συμπληρωμα του ενος (ΣΤ1)	~0x0003	0xfffc
>>	Μετατοπιση (shift) δεξια	0x7f >> 2	0x1f
<<	Μετατοπιση (shift) αριστερα	0x1f << 2	0x7c
&	Λογικο και (AND)	0x8a & 0x7f	0x0a
^	Αποκλειστικο ή (XOR)	0x8A ^ 0xC3	0x3C
|	Λογικο ή (OR)	0x42 | 0x36	0x76

Τελεστες πραξεων και απονομης τιμης:

```
>>=     <<=     &=     ^=     |=
```

πχ. `a >>= 4` Ιδιο με: `a = a >> 4;`

Παραδειγματα

```
a) char ch = 1;
```

Δεκαεξαδικο:				0				1
Δυαδικο :	0	0	0	0	0	0	0	1

```
   ch = ch << 4;  // Μετατοπιση 4 θεσεις προς αριστερα
```

Δεκαεξαδικο:				1				0
Δυαδικο :	0	0	0	1	0	0	0	0

```
   cout << ch ;   // Εμφανιση 16
```

```
b) unsigned char ch = 0xFF;
```

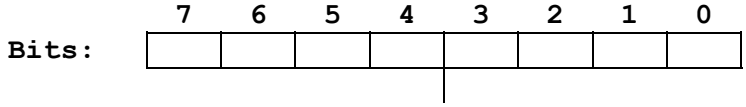
Δεκαεξαδικο:				F				F
Δυαδικο :	1	1	1	1	1	1	1	1

```
   ch = ch >> 4;  // Μετατοπιση 4 θεσεις προς δεξια
```

Δεκαεξαδικο:				0				F

```
   Δυαδικο    :  | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

   cout << ch;   // Εμφανιση 15
---
```

Δεκαεξαδικες τιμες που ειναι χρησιμες στον ορισμο σταθερων (με bits ισα με 1):



Bit = 1	Τιμη
0	0x01
1	0x02
2	0x04
3	0x08
4	0x10
5	0x20
6	0x40
7	0x80

Εργασιες με bits

Συνηθεις εργασιες με bits ειναι:

α) Θετουμε την τιμη 1 σε ενα bit (set a bit). πχ.
```
   c = c | 0x80;  // το bit 7 παιρνει την τιμη 1
```
β) Μηδενιζουμε την τιμη σε ενα bit (clear a bit) πχ.
```
   c = c ^ 0x01;  // το bit 0 παιρνει την τιμη 0
```
γ) Ελεγχος τιμης ενος bit (test a bit) πχ.
```
   if( (c & 0x40) == 0 ) // ελεγχος τιμης του bit 6
```

```
#define READ   0x01
#define WRITE  0x10

int main()
{
    cout << hex << READ | WRITE << endl; // 0x11
    return(0);
}
```

-- Να γραφει ενα προγραμμα το οποιο διαβαζει εναν char (απο 0-255) και στην συνεχεια εμφανιζει τον αριθμο ως bit-πινακα (με 0 και 1)

```
#include <iostream.h>
int main()
{
    char num; int i, t, z1;
    cin >> num;
    t = 0x80;
    for(i = 0; i < 8; i++) {
        z1 = num & t;
        num = mun << 1;
        z1 = z1 >> 7;
        cout << z1;
    }
```

```
    return(0);
}
```

-- Να γραφει μια macro bitmask η οποια εκτελει bitwise AND στο περιεχομενο δυο μεταβλητων x και y. (πχ. **0x0001 & 0x0011 -> 0x0001**)

Λυση

```
#define  bitmask(x,y)    (x & y)
```

-- Να γραφει μια συναρτηση xor η οποια εκτελει exclusive or στα bits των περιεχομενων των μεταβλητων x και y. Ο πινακας αληθειας για το exclusive or ειναι:

X	Y	Αποτελεσμα
0	0	0
0	1	1
1	0	1
1	1	0

Βοηθημα:

1. Ανεστρεψε την τιμη του y και εκτελεσε bitwise AND με την τιμη του x.
2. Ανεστρεψε την τιμη του x και εκτελεσε bitwise AND με την τιμη του y.
3. Εκτελεσε ενα bitwise OR στα αποτελεσματα των δυο παραπανω παραστασεων.

Λυση

```
#include <iostream.h>
int xor(int, int);
int main(void)
{
    int a,b,r;
    a = 3;
    b = 5;
    r = xor(a, b);
    cout << hex;
    cout << "XOR του " << a << " και του " << b
         << " ειναι = " << r << endl;
    return(0);
}

int xor(int x, int y)
{
    return((x & ~y) | (~x & y))
}
```

Αποτελεσμα

```
XOR του 0x0011 και του 0x0101 ειναι = 0x0110
```

Αν η **xor** γραφει ως macro τοτε εχει την μορφη:

```
#define xor(x,y)  ((x & ~y) | (~x & y))
```

-- Να εμφανισετε ενα πινακα με τις δυναμεις του 2. Απο το 2 εις την 0 εως 2 εις την 15 (ή 31) δηλ.:

```
2 εις την 0  1
2         1  2
2         2  4
   ...κοκ...
```

Βοηθημα: εαν ο int ειναι 2 bytes τοτε η μεγαλητερη δυναμη του 2 ειναι ((2*8) - 1); χρησιμοποιειστε αριστερη μετατοπιση σε μια μεταβλητη που η αρχικη της τιμη ξεκινα απο 2. (bit no 1 = 1 ==> 2 εις την 1 = 2)

```
#include <iostream.h>
int main()
{
    unsigned int i;  int ek,tb;
    tb = 8*sizeof(int);    // Πληθος των bits σε εναν int
    cout << "Πινακας των δυναμεων του 2\n";
    cout << "--------------------------\n";
    // το 2 εις την 0 μηδενικη δυναμη = 1
    cout << "2 εις την  0   1" << endl;
    i = 1;  // τοποθετηση του 1 στο πρωτο bit;
    for(ek = 1; ek <= tb - 1; ek++) {
        i = i << 1;    //  Μετατοπιση του 1 προ τα αριστερα
                       //  κατα μια θεση
        cout << "2 εις την " << ek << "   " << i << endl;;
    }
    return(0);
}
// p19.cpp //
```

Αποτελεσμα

```
Πινακας των δυναμεων του 2
--------------------------
2 εις την  0  1
2 εις την  1  2
2 εις την  2  4
2 εις την  3  8
. . . . . . . . . .
2 εις την 13  8192
2 εις την 14  16384
2 εις την 15  32768
```

Αναδρομη (recursion)

Οταν μια συναρτηση καλει (κανει κληση) τον εαυτο της τοτε εχουμε αναδρομη (recursion).

-- Να γραφει μια συναρτηση power η οποια χρησιμοποιει recursion (η συναρτηση καλει τον εαυτον της) και υπολογιζει το x υψωμενο εις την y (x^y).

```
#include <iostream.h>
int power(int,int);

int main(void)
{
    cout << power(2,15) << endl;     // 2^15
    return(0);
}

int power(int x,int y)
{
    if (y == 0)
        return(1);
    else {
        y--;                 // πχ. αντι 5^3 -> 5*5^2
        x = x * power(x,y);
    }
    return(x);
} // ask30.cpp
```

Αποτελεσμα

```
32768
```

-- Να γραφει μια συναρτηση revstr η οποια εμφανιζει ενα string με αντίστροφη σειρα δηλ. ο τελευταιος χαρακτηρας θα εμφανισθει πρωτος. Να χρησιμοποιειθει αναστροφη (recursion).

```
#include <iostream.h>
void revstr(char *);
int main(void)
{
    revstr("Qwerty");
    return(0);
}

void revstr(char *s)
{
    if(*s == '\0')
        return;
    else {
        revstr(++s);
        cout << *(--s);
    }
} // ask31.cpp
```

Αποτελεσμα

```
ytrewQ
```

-- Να γραφει μια συναρτηση (με αναδρομη) η οποια επιστρεφει για εναν n το:

```
n + n-1 + n-2 + ... + 2 + 1
```

Παραμετροι Γραμμης-Εντολων (Command-Line arguments)

Η γραμμη εντολων ειναι η γραμμη οπου δινουμε εντολη να εκτελεσθει το προγραμμα μας (αφου βεβαιως το εχουμε μεταφρασει). (Η γραμμη εντολων μπορει επισης να ειναι ενα batch αρχειο στο DOS ή ενα script στο UNIX) Η πρωτη λεξη στην γραμμη εντολων ειναι το εκτελεσιμο αρχειο, οτιδηποτε λεξεις (string) ακολουθουν (εκτος απο αλλαγη-διευθυνσης αρχειου file-redirection) συγκεντρωνονται και παιρνούν ως παραμετροι στο main(). Ετσι οι παραμετροι της γραμμης-εντολων ειναι strings που περνουν απο την γραμμη-εντολων στο **`main()`**. Η δηλωση των παραμετρων-γραμμης-εντολων γινεται ως εξης:

```
int main(int argc, char *argv[])
// argc = Μετρητης παραμετρων
// argv = Δεικτες στον πινακα των παραμετρων
{
    // εντολες
}
```

Παραδειγμα: **`ask33.cpp`**

Το προγραμμα εμφανιζει το πληθος των παραμετρων και τις παραμετρους καθως και τους δευτερους χαρακτηρες των παραμετρων.

```
#include <iostream.h>
int main(int ac, char *av[])
{
    int i;
    cout << "Πληθος παραμετρων = " << ac << endl;
    // Εμφανιση παραμετρων
    for(i = 0; i < ac; i++)
        cout << av[i] << "   "    // Εμφ. της παραμετρου
             << av[i][1] << endl; // Εμφ. του 2ου χχαρακτηρα
    return(0);
}
```

Εαν στην γραμμη εντολων γραψουμε:

```
ask33.exe dog elephant tiger
```

Τοτε θα εχουμε το εξης αποτελεσμα:

```
Πληθος παραμετρων = 4
ask33.exe  s
dog        o
elephant   l
tiger      i
```

Ολες οι παραμετροι περνουν ως string. Ετσι εαν θελουμε να περασουμε αριθμους πρεπει να τους μετατρεψουμε μεσα στο προγραμμα μας απο string σε αριθμους. Μπορουμε να χρησιμοποιησουμε τις standard συναρτησεις atoi, atof, atol ή την συναρτηση vall (βλεπε παραπανω)

Δομες (`struct`) και Ταξεις (`class`)

Οι δομες που οριζονται με **`struct`** ή με **`class`** ειναι ισοδυναμες στην C++. Η μονη διαφορα ειναι οτι τα μελη σε μια **`struct`** δομη ειναι εξ' ορισμου δημοσια (**`public`**) ενω τα μελη σε μια class ειναι εξ'ορισμου ιδιωτικα (**`private`**). Συνηθως για την δημιουργια αντικειμενων σε αντικειμενοστρεφη προγραμματα χρησιμοποιειται η **`class`**. Τα δεδομενα (απλοι τυποι οπως ακεραιοι, πραγματικοι, δεικτες κλπ, ή συνθετοι τυποι οπως πινακες, δομες κλπ) και συναρτησεις που δηλωνονται μεσα σε μια struct ή class ονομαζονται μελη (members).

Τα μελη (μεταβλητες και συναρτησεις) μιας struct ή μιας class μπορουν να ορισθουν (οσο αφορα τον τροπο πρόσβασής των) με εναν απο τους παρακατω τρεις τροπους:
a) `private` (ιδιωτικα). Τα ιδιωτικα μελη της ταξης ειναι προσπελασιμα μονο απο συναρτησεις μελη της ταξης.
b) `public` (δημοσια). Τα δημοσια μελη της ταξης ειναι προσπελασιμα απο ολο το προγραμμα.
c) `protected` (προστατευομενα). Τα προστατευομενα μελη της ταξης ειναι προσπελασιμα μονο απο συναρτησεις μελη. (Ειναι οπως τα ιδιωτικα αλλα διαφερουν οταν κληρονομουνται)

Δομες (structures)

Ο τυπος δεδομενων δομη (struct) μας επιτρεπει να συνδιασουμε δεδομενα απλων τυπων και πινακες πολλων τυπων κατω απο το ιδιο ονομα. Η C/C++ μας επιτρεπει να απλοποιησουμε τον χειρισμο δομων με μεγαλο πληθος στοιχειων οργανωνοντας τις δομες σε υποδομες. Οι δομες μοιαζουν με εγγραφες (records) και ειναι καταλληλες για διαχειριση των δεδομενων στην μνημη.

Ορισμος δομης:

```
    struct ονομα_δομης  {
         δηλωση_στοιχειων_δομης
    };
```

Τα *στοιχεια_δομης* ειναι μεταβλητες ή συναρτησεις μελη της δομης.
Παραδειγματα

```
α) struct A {
       int x;     // Δημοσια μελη: x και ch
       char ch;
   };

β) struct B {
       int getx();  // Δημοσια συναρτηση μελος getx
   private:
       int x;       // Ιδιωτικο μελος x
   };

γ) struct C {
       double f1;  // Δημοσιο μελος
   private:
       int a;      // Ιδιωτικο μελος a
       int geta(); // Ιδιωτικη συναρτηση geta
```

```
protected:
    int b;       // Προστατευομενο μελος
public:
    void setx(int m); // Δημοσια συναρτηση μελος setx
};
```

Δηλωση μεταβλητων ή αντικειμενα δομης:

```
struct ονομα_δομης μεταβλητη_δομης;
ή
ονομα_δομης μεταβλητη_δομης;
```

Ο ορισμος μιας δομης Disp:

```
struct Disp {            // Ονομα δομης Disp
    int line;
    int column;
    char *message;
};
```

Τωρα η δομη Disp ειναι ενας "καινουργιος" τυπος δεδομενων, ετσι μπορουμε να δηλωσουμε μεταβλητες ή αντικειμενα του τυπου Disp ως εξης:

```
α) struct Disp A; // Μεταβλητη ή αντικειμενο A ειναι τυπου Disp
   ή
   Disp A;        // Αντικειμενο A τυπου Disp

β) struct Disp *ptrDisp;  // Η μεταβλητη ptrDisp ειναι δεικτης σε
   ή Disp *ptrDisp;          δομη τυπου Disp

γ) Disp B = {
                4,
               10,
               "Ονομα Πελατη"
           };
```

Δηλωνεται το αντικειμενο **B** τυπου **Disp**, και διδονται αρχικες τιμες στις μεταβλητες του **B**

Προσβαση σε μεταβλητες δομης:

```
αντικειμενο_δομης.στοιχειο_δομης
```

Παραδειγμα απονομη τιμων

```
A.line = 3;
A.column = 15;
A.message = "Clown";
```

Παραδειγμα ανακτησης τιμων

```
cout << B.line << endl;
cout << B.message << endl;
```

Ο προηγουμενος ορισμος της δομης Disp μπορει να ορισθει ως δυο δομες η μια μεσα στην αλλη (**nested structures**):

```
struct cur_loc {
    int line
    int column;
};

struct Disp {
    struct cur_loc cursor;
    char *message;
};
```

Εαν δηλωσουμε μια μεταβλητη Z του τυπου struct Disp τοτε μπορουμε να εχουμε προσβαση στα στοιχεια της δομης ως εξης:

```
Disp Z;
:
Z.cursor.line = 15;
Z.cursor.column = 29;
Z.message = "Ha-ha said the clown";
```

Εαν ορισθει ενας δεικτης σε μια δομη τοτε χρησιμοποιουμε τον τελεστη **->** για να εχουμε προσβαση στις μεταβλητες της δομης.

```
Disp Z,          Αντικειμενο Z τυπου Disp
     *ptrDisp;   Δηλωση του δεικτη.
ptrDisp = &Z;    Ο δεικτης δειχνει στο αντικειμενο Z.
```

Τοτε με **ptrDisp->*μελος*** ή **(*ptrDisp).*μελος*** εχουμε προσβαση σε στοιχεια της δομης Disp.

```
ptrDisp->line = 12;   // ιδιο με: (*ptrDisp).line = 12
cout << ptrDisp->line << endl;
cout << (*ptrDisp).message << endl;
```

-- Παραδειγμα απονομης και ανακτησης τιμων στοιχειων (πεδίων) δομης.

```
#include <iostream.h>
// Εγγραφη σπουδαστη.
struct spoud {
    int   asm;        // Αριθμος σπουδαστικου μητρωου.
    char  name[30];   // Ονομα σπουδαστη.
    char  sex;        // Φυλο σπουδαστη ('M' ή 'F').
    float grad;       // Βαθμος στο μαθημα C++.
};

int main(void)
{
    struct spoud spd; // Δηλωση μεταβλητης spd.
    // Εισαγωγη στοιχειων.
    cout << "Εισαγωγη Στοιχειων Σπουδαστη" << endl;
    cout << "Αριθμ. Σπουδ. Μητρ. ";
    cin >> spd.asm;

    cout << "Ονομα Σπουδαστη.... ";
    cin >> spd.name;
    cout << "Φυλλο (M/F) ..... ";
    cin >> spd.sex;
    cout << "Βαθμος στη C++ .... ";
    cin >> spd.grad;
```

```
    // Εμφανιση στοιχειων.
    cout << "Εμφανιση Στοιχειων Σπουδαστη" << endl;
    cout << "Αριθμ. Σπουδ. Μητρ. " << spd.asm << endl;
    cout << "Ονομα Σπουδαστη.... " << spd.name << endl;
    cout << "Φυλλο (M/F) ....... " << spd.sex << endl;
    cout << "Βαθμος στη C++ .... " << spd.grad << endl;
    return(0);
}
```

Παραδειγμα πινακα μεσα σε δομη

```
   struct A {          // Ορισμος δομης
      int ar[10];      // Δηλωση πινακα ar
      char ch;
      double m;
   }
   :
   A a;               // Δηλωση αντικειμενου a
   :
   a.ar[0] = 23;      // Απονομη τιμης σε κελλι του πινακα a
   :
   cout << a.ar[4];   // Ανακτηση τιμης απο κελλι του πινακα a
---
```

Το **<u>περασμα δομων σε συναρτησεις</u>** γινεται οπως γινεται σε ολους τους απλους τυπους δεδομενων της C++.

-- Παραδειγμα: Περασμα δομης ως παραμετρο σε συναρτηση. Το προγραμμα εισαγει και εμφανιζει στοιχεια για δυο σπουδαστες.

```
#include <iostream.h>
// Εγγραφη σπουδαστη.
struct Spoud {
    int   am;         // Αριθμος σπουδαστικου μητρωου.
    char  name[30];   // Ονομα σπουδαστη.
    char  sex;        // Φυλο σπουδαστη ('M'/'F').
    float grad;       // Βαθμος στο μαθημα C++.
};

void eisagogi(Spoud *);
void emfanisi(Spoud);
void emfapart(Spoud &z); // Μερικη εμφανιση

int main()
{
    Spoud spd,        // Δηλωση αντικειμενο δομης spd
          *ps;        // Δεικτης-δομης ps.

    eisagogi( &spd );// Κληση με διεθυνση (call by reference)
                     // διοτι απο την συναρτηση θα αλλαξουν
                     // τα στοιχεια του αντικειμενου spd

    emfanisi( spd );   // Κληση με την τιμη της δομης
                       // (call by value)
```

```
    emfapart( spd );   // Κληση με την τιμη της δομης
                       // (call by reference)

    ps = &spd;         // Ο δεικτης δειχνει τωρα στο spd.

    eisagogi( ps );    // call by reference με χρηση δεικτη

    emfanisi( spd );   // Κληση με την τιμη της δομης
                       // (call by value)
    return(0);
}

void eisagogi(Spoud *x)
{
    cout << "============================" << endl;
    cout << "Εισαγωγη Στοιχειων Σπουδαστη" << endl;
    cout << "Αριθμ. Σπουδ. Μητρ. ";
    cin >> x->am;

    cout << "Ονομα Σπουδαστη.... ";
    cin >> x->name;

    cout << "Φυλλο (M/F) ..... ";
    cin >> x->sex;

    cout << "Βαθμος στη C++ .... ";
    cin >> x->grad;
}

void emfanisi(Spoud z)
{
    cout << endl << "------" << endl;
    cout << "Εμφανιση στοιχειων Σπουδαστη "   << endl;
    cout << "Αριθμ. Σπουδ. Μητρ. " << z.am   << endl;
    cout << "Ονομα Σπουδαστη.... " << z.name << endl;
    cout << "Φυλλο (M/F) ..... " << z.sex  << endl;
    cout << "Βαθμος στη C++ .... " << z.grad << endl;
}

void emfapart(Spoud &z)
{
    cout << endl << "------" << endl;
    cout << "Εμφανιση Βαθμολογιας Σπουδαστη "   << endl;
    cout << "Ονομα Σπουδαστη.... " << z.name << endl;
    cout << "Βαθμος στη C++ .... " << z.grad << endl;
}
// p28.cpp //
```

Πινακες δομων

Ο ορισμος ενος πινακα δομων (arrays of structures) γινεται ως εξης:

```
α) struct ονομα_δομης ονομα_μεταβλητης[αριθμος_κελλιων];
β) ονομα_δομης ονομα_μεταβλητης[αριθμος_κελλιων];
```

Παραδειγματα:

```
α)  struct term ardom[4];
β)  Stud cpp_Lab[20];
γ)  Disp array_Mes[4] = {
        3, 10, "Ονομα πελατη",
        4, 10, "Διευθυνση",
        5, 10, "Τηλεφωνο",
        0,  0, "\0"          // Null_char
    };
```

Η προσβαση μεταβλητης (στοιχειου) σε πινακα δομων γινεται ως εξης:

```
ονομα_πινακα[αριθμος_κελλιου].στοιχειο

a) cout << array_Mes[2].line; // Εμφανιζεται: 5

b) cout << array_Mes[1].message; // Εμφανιζεται: Διευθυνση

c) strcpy(cpp_Lab[0].name,"Adam"); // Απονομη τιμης
```

Το **περασμα πινακα-δομων σε συναρτησεις** γινεται οπως γινεται με τους πινακες "απλων" αντικειμενων (int, char, float κλπ) :

α) Με γραφη πινακα

```
void screen(Disp []);     // Πρωτοτυπο
:
Disp ar[6];
:
screen(ar);               // Κληση
:
void screen(Disp pin[])   // Επικεφαλιδα
{
    ...
}
```

β) Με δεικτη

```
void screen(Disp *);    // Πρωτοτυπο
:
Disp a[10];
:
screen(a);              // Κληση
:
void screen(Disp *pd)   // Επικεφαλιδα
{
    ...
{
```

-- Παραδειγμα περασματος πινακα δομων ως παραμετρο σε συναρτηση. Το προγραμμα εισαγει και εμφανιζει στοιχεια για τρεις σπουδαστες.

```
#include <iostream.h>

#define NUM_STUD  3  // Πληθος σπουδαστων

// Εγγραφη σπουδαστη.
struct Spoud {
    int   am;          // Αριθμος σπουδαστικου μητρωου.
    char  name[30];    // Ονομα σπουδαστη.
    char  sex;         // Φυλο σπουδαστη ('M'/'F').
    float grad;        // Βαθμος στο μαθημα C++.
};

void eisagogi( Spoud * );
void emfanisi( Spoud [] );

int main(void)
{
    Spoud ar_spd[NUM_STUD];  // Δηλωση πινακα δομων

    eisagogi( &ar_spd[0] );  // Κληση με διευθυνση
                             // (call by reference)
    emfanisi(ar_spd);//Κληση με διευθυνση (call by reference)

    return(0);
}

void eisagogi( Spoud *x )
{

    for(int i=0; i < NUM_STUD; i++) {
        cout << "============================" << endl;
        cout << "Εισαγωγη Στοιχειων Σπουδαστη No "
             << i+1 << endl;
        cout << "Αριθμ. Σπουδ. Μητρ. ";
        cin >> (x+i)->am;

        cout << "Ονομα Σπουδαστη.... ";
        cin >> (x+i)->name;              // A

        cout << "Φυλλο (M/F) ....... ";
        cin >> x[i].sex;                 // B

        cout << "Βαθμος στη C++ .... ";
        cin >> (*(x+i)).grad;            // Γ
    }
}

void emfanisi( Spoud z[])
{
    int i = 0;

    for( ; i < NUM_STUD; i++) {
        cout << endl << "------" << endl;
        cout << "Εμφανιση στοιχειων Σπουδαστη No "
```

```
             << i+1 << endl;
        cout << "Αριθμ. Σπουδ. Μητρ. " << z[i].am << endl;
        cout << "Ονομα Σπουδαστη.... " << z[i].name << endl;
        cout << "Φυλλο (M/F) ....... " << (*(z+i)).sex
             << endl;
        cout << "Βαθμος στη C++ .... " << (z+i)->grad
             << endl;
    }
}
// p30.cpp //
```

Προσεξετε στα σημεια **Α**, **Β**, **Γ** την διαφορετικη γραφη προσβασης σε πινακα-δομων.

Bit-Πεδια (bit fields)

Τα πεδια-bit δηλωνονται οπως μια δομη

```
struct ονομα_δομης_πεδιων_bits {
    τυπος ονομα_πεδιου1 : μηκος;
    τυπος ονομα_πεδιου2 : μηκος;
          ...
    τυπος ονομα_πεδιουN : μηκος;
} μεταβλητες;
```

Το αριθμος των bits σε ενα πεδιο-bits οριζεται απο το *μηκος*.

Η **προσβαση σε πεδια-bit** γινεται οπως γινεται σε μελη δομης (στοιχεια δομης)

```
α)  struct PeD {            // Ορισμος δομης πεδιων-bits
        unsigned b1 : 1;
        unsigned b2 : 1;
        unsigned b3 : 1;
    };

    PeD m, *p;      // Μεταβλητη m δομης πεδιων-bits
    p = &m;

    m.b1 = 1;       // Απονομη τιμης σε πεδιο-bit
    p->b2 = 0;      // Με χρηση δεικτη

    cout << m.b2;   // Ανακτηση τιμης απο πεδιο-bit
    cout << p->b3;

β)  struct BitFi {
        unsigned :    4;    // Πεδιο χωρις ονομα με 4 bits
        unsigned rd : 1;
        unsigned wr : 1;
    };

γ)  struct BFi {
        struct BitFi ma;
        float f;
        unsigned qv : 1;
        unsigned dts: 1;
    };
```

-- Παραδειγμα

```
#include <iostream.h>

struct BF {
   unsigned b0 : 1;
   unsigned b1 : 1;
   unsigned b2 : 1;
   unsigned b3 : 1;
};
```

```
int main(void)
{
   BF A;

   A.b0 = 1;
   A.b1 = 1;

   cout << A.b0 << endl;
   cout << A.b1 << endl;

   if(A.b0 == 1)
      cout << "OK!" << endl;
   else
      cout << "Boo!" << endl;

   return(0);
}
// g90.cpp //
```

Ενωση (union)

Μια **union** ειναι ενα κοματι μνημης που μοιραζεται απο δυο ή και περισσοτερες διαφορετικες μεταβλητες.

Ορισμος ενωσης:

```
union ονομα_ενωσης  {
      δηλωση_στοιχειων_ενωσης
};
```

Παραδειγμα

Ορισμος ενωσης utyp:

```
union utyp {
    int i;
    char ch;
};
```

Ορισμος μεταβλητων τυπου utyp:

```
union utyp x,y;
```

Προσβαση στα στοιχεια της ενωσης:

```
x.i = 10;
```

Εαν υποθεσουμε οτι στον υπολογιστη μας ενας int καταλαμβανει 2-bytes τοτε για την ενωση utyp εχουμε:

```
--- ch ---

| Byte 0 | Byte 1 |

--------- i ---------
```

```
#include <iostream.h>
union utyp {
    int i;
    char ch;
```

```
};

int main()
{
    union utyp x;
    x.i = 65;   // 65 = 'A' στον πινακα ASCII
    cout << "Τιμη του i  = " << x.i << endl;

    // Επειδη ο i και ο ch μοιραζονται την ιδια μνημη
    // ο ch παιρνει την τιμη του i.

   cout << "Τιμη του ch = " << x.ch << endl;
    return(0);
}
// p46.cpp
```

Αποτελεσμα

```
Τιμη του i  = 65
Τιμη του ch = A
```

Απαριθμητοι τυποι (Enumerations)

Απαριθμητος ή βαθμοτος τυπος (enumeration) ειναι ενα συνολο απο ακεραιους που οριζει ολες τις επιτρεπομενες τιμες που μπορει να εχει μια μεταβλητη αυτου του τυπου. Οριζεται με την λεξη κλειδι **enum**.

```
enum ονομα_απαριθμητου_τυπου {
        ονοματα
};
```

Παραδειγματα

α) `enum imera {dey, tri, teta, pem, para, sabbato, kyriaki};`
Η **dey** εχει τιμη 0, η **tri** εχει τιμη 1 κ.ο.κ.
`cout << dey << " " << kyriaki ;`
Θα εμφανισθει 0 6

β) `enum zestoi_mines {maios = 5, ioun, ioul, avgoust, sept};`
Η αριθμηση αρχιζει απο το 5, ετσι ο **ioun** λαμβανει την τιμη 6, ο **ioul** 7 κ.ο.κ.
`cout << maios << " " << avgoust ;`
Θα εμφανισθει 5 8

γ) Η συγκριση **tri < sabbato** ειναι αληθης (συμφωνα με το προηγουμενο **enum imera)**

-- Στο παρακατω προγραμμα για να εμφανισθει κατι στην οθονη πρεπει ο χρηστης να δωσει τιμες μια απο 0,1,2,3.

```
#include <iostream.h>
enum xroma { kok, kit, ble, pras };
main()
{
    enum xroma db;
```

```
    cout << "Δωσε μια τιμη απο 0,1,2,3 ";
    cin >> db;
    switch(db) {
        case kok :  cout << "Κοκκινο\n"; break;
        case kit :  cout << "Κιτρινο\n"; break;
        case ble :  cout << "Μπλε\n";    break;
        case pras : cout << "Πρασινο\n"; break;
    }
    return(0);
}
// p47.cpp //
```

-- Το ιδιο προγραμμα με διαφορετικο κωδικα.

```
#include <iostream.h>

char nam [][12] = {
        "Κοκκινο",
        "Κιτρινο",
        "Μπλε",
        "Πρασινο"
};

enum xroma { kok, kit, ble, pras };

int main()
{
    enum xroma db;
    cout << "Δωσε μια τιμη απο 0,1,2,3 ";
    cin >> db;
    if(db >= kok && db <= pras)  // Ειναι εντος οριων;
        cout << nam[db] << endl;
    return(0);
}
// p48.cpp //
```

Ασκησεις

-- Η δομη spoud να εμπλουτισθει με εναν δεικτη τυπου spoud και να δηλωθουν οι μεταβλητες **st1**, **st2**, ***pst**, **ars[4]** ολες τυπου δομης **Spoud.**

```
struct Spoud {
    int   am;
    char  name[30];
    char  sex;
    float grad;
    struct Spoud *sptr;
} st1, st2, *pst, ars[4];
```

-- Να γραφει ο κωδικας που δημιουργει δυναμικα (κατα την εκτελεση του προγραμματος) μια δομη (εγγραφή) τυπου spoud. Δηλωσετε εναν δεικτη να δειχνει στη δομη.

```
Spoud *rec;    // Δηλωση δεικτη.
rec = new Spoud;
```

-- Να γραφει ενα προγραμμα το οποιο δημιουργει μια δυναμικη λιστα με πεντε εγγραφες τυπου spoud. Σε καθε στοιχειο της λιστας θα εισαχθουν δεδομενα για εναν σπουδαστη. Η λιστα να μη επιτρεπει την δημιουργια περισσοτερες απο πεντε εγγραφες.

```
#include <iostream.h>
#define  NIL  0L
struct spoud {
    int   am;
    char  name[30];
    char  sex;
    float grad;
    struct spoud *sptr;
};
typedef struct spoud STUD;
int main(void)
{
    STUD *p1,  // Δεικτης στο πρωτη εγγραφη
         *pC,  // Δειτης στην τρεχουσα εγγραφη
         *pT;  // Βοηθητικος δικτης
    p1 = new spoud;
    pT = p1;
    eisagogi( p1 );
    while(1) {
        pC = new spoud;
        eisagogi( pC );
        pT->sptr = pC;
        pT = pC;
    }
    return(0)
}

void eisagogi(struct spoud *x)
{
    cout << "Εισαγωγη Στοιχειων Σπουδαστη" << endl;
    cout << "Αριθμ. Σπουδ. Μητρ. ";
    cin >> x->am;
```

```
    cout << "Ονομα Σπουδαστη.... ";
    cin >> x->name;
    x->sptr = NIL;
}
```

-- Να γραψετε ενα προγραμμα το οποιο να δημιουργει, να εμφανιζει, και διαγραφει μια λιστα με "στοιχεια" (items, records) δομης:

```
    struct stud {
        int aa;
        char name[20];
        stud *pS;    // Pointer στην δομη.
    };
```

Λυση

```
#include <iostream.h>

// Ορισμος δομης
struct stud {
    int aa;
    char name[20];
    stud *pS;    // pointer στην δομη
};

typedef struct stud STUD;

// Πρωτοτυπα συναρτησεων
STUD *dimlist();
STUD *eisagogi();
void emfanisi(STUD *);
void diagrafi(STUD *);

int main(void)
{
    STUD *pa;   // Δεικτης στη δομη.

    pa = dimlist();  // Ο pa δειχνει στην πρωτη εγγραφη
    if (pa == 0)     // Εαν ο pa ειναι 0 δεν υπαρχει λιστα
        return(0);  // και τελος προγραμματος.
    emfanisi( pa );  // Εμφανιση λιστας
    diagrafi( pa );  // Διαγραφη της λιστας

    return(0);
}

// Διαγραφη (καταστροφη της λιστας)
// Επιστρεφει:  void
void diagrafi( STUD *p )
{
    STUD *nx;
    while( p ) {   // Εκτελεση θηλειας, οσο ο p δειχνει
                   // σε εγγραφη.

       nx = p->pS; // Πριν διαγραφη η εγγραφη σωσε τον δεικτη
                   // που δειχνει στην επομενη εγγραφη.
                   // Ο nx δειχνει στην επομενη εγγραφη.
```

```
       delete p;   // Διαγραφη εγγραφης

       p = nx;     // Ο p δειχνει στην επομενη εγγραφη.
    }
}

// Εμφανιση της λιστας.
// Επιστρεφει: - void
void emfanisi(STUD *p)
{
    while( p ) {
        cout << p->aa   << endl;
        cout << p->name << endl;
        cout << "-----" << endl;
        p = p->pS;
    }
}

// Δημιουργια λιστας.
// Επιστρεφει: - εναν δεικτη στην πρωτη εγγραφη της λιστας
//             - 0 εαν η λιστα ειναι κενη εγγραφων.
STUD *dimlist()
{
    STUD *pa,  // Δεκτες: pa πρωτης εγγραφης
         *pt,  //         pt τελευταιας  εγγραφης
         *pc;  //         pc τρεχουσας     ..
    pa = pt = pc = eisagogi(); // Ολοι οι δεικτες δειχνουν
                               // στην πρωτη εγγραφη.
    if (pc == 0)    // Ακυρωση δημιουργιας λιστας ?
        return(0);  // εαν ναι Τελος.
    while(1) {
        pc = eisagogi();  // Εισαγωγη δεδομενων στην εγγραφη
        if ( pc == 0 )    // Τελος εισαγωγης ?
            break;
        pt->pS = pc;      // Συνδεση εγγραφης με την λιστα
                          // (ο δεικτης δομης της τελευταιας
                          // εγγραφης της λιστας δειχνει
                          // στην τρεχουσα εγγραφη.
        pt =pc;     // Η τρεχουσα εγγραφη γινεται
                    // τελευταια της λιστας.
    }
    return(pa);
}

// Δημιουργια και εισαγωγη δεδομενων εγγραφης.
// Επιστρεφει: - Δεικτη στην εγγραφη
//             - 0 (μηδεν) εαν ειναι τελειωσε η εισαγωγη των
//               στοιχειων των σπουδαστων.
STUD *eisagogi()
{
    STUD *p;
    p = new STUD; // Δημιουργια εγγραφης
    cout << "ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΩΝ ΣΠΟΥΔΑΣΤΗ" << endl;
    cout << "Αυξων Αριθμος (0=τελος) "; cin >> p->aa;
```

```
    if ( (*p).aa == 0 ) {
        delete p;    // Διαγραφη της εγγραφης
        return(0);   // Τελος εισαγωγης εγγραφων.
    }
    cout << "Ονομα........";   cin >> p->name;
    p->pS = 0;  // Ο δεικτης δομης δεν δειχνει πουθενα.
    return(p);  // Επιστροφη του δεικτη στην εγρραφη.
}
// p31.cpp //
```

-- Στην προηγουμενη ασκηση προσθεσετε μια συναρτηση η οποια επιστρεφει το πληθος των εγγραφων της λιστας.

-- Στην προηγουμενη ασκηση προσθεσετε μια συναρτηση η οποια για εναν αυξοντα αριθμο (εαν υπαρχει στην λιστα) εμφανιζει στα στοιχεια της εγγραφης αλλιως εμφανιζει αναλογο μηνυμα.

-- Στην προηγουμενη ασκηση προσθεσετε μια συναρτηση η οποια για εναν αυξοντα αριθμο (εαν υπαρχει στην λιστα) καθιστα την εγγραφη ανενεργη (δωσε τιμη aa = -1) (δηλ. λογικη διαγραφη της εγγραφης και οχι φυσικη που γινεται με τον τελεστη **delete**) αλλιως εμφανιζει αναλογο μηνυμα.

-- Στην προηγουμενη ασκηση προσθεσετε μια συναρτηση η οποια προσθετει μια εγγραφη στην λιστα με την εξης λογικη: εαν υπαρχει εγγραφη ενενεργη/μηδενισμενη (aa = -1) τοτε τα στοιχεια της νεας εγγραφης καταλαμβανουν το χωρο της μηδενισμενης εγγραφης (δηλ. η εγγραφη καθισταται ενεργη) αλλιως η νεα εγγραφη προστηθεται στο τελος της λιστας.

-- Να γραψετε ενα προγραμμα το οποιο θα δημιουργει, θα εμφανιζει τις εγγραφες με την σειρα που εισηχθησαν και αναστροφα, και τελος θα καταστρεφει την λιστα με εγγραφες της παρακατω δομης. (Για να γινει εμφανιση εγγραφων αναστροφα θα χρησιμοποιειθει ενας ακομη δεικτης για να δειχνει την προηγουμενη εγγραφη)

```
    struct Spoud {
        int   am;
        char  name[30];
        char  sex;
        float grad;
        Spoud *epo;  // Δεικτης στην προηγουμενη εγγραφη.
        Spoud *pro;  // Δεικτης στην επομενη εγγραφη.
    };
```

-- Να γραψετε ενα προγραμμα το οποιο θα δημιουργει, θα εμφανιζει και τελος θα καταστρεφει την λιστα με εγγραφες της παρακατω δομης. Η λιστα θα ειναι ταξινομημενη κατα αυξουσα διαταξη ως προς το πεδιο aab, αυτο σημαινει οτι οι εγγραφες θα εισαγονται σε οποιοδηποτε σημειο της λιστας.

```
    struct Book {
        int aab;
        char author[15];
        char title[50];
        float price;
        Book *pb;
    };
```

-- Να γραψετε ενα προγραμμα το οποιο θα δημιουργει, θα εμφανιζει και τελος θα καταστρεφει μια συνθετη λιστα με εγγραφες των παρακατω δομων book (βιβλιο) και stud (σπουδαστης). Η λιστα κρατα στοιχεια για το ποσα και ποια βιβλια εχει δανιστει ενας σπουδαστης. Ο σπουδαστης μπορει να δανιστει απο 0 εως 5 βιβλια.

```
struct Book {
    int kb;           // Κωδικος βιβλιου
    char dat[9];      // Ημερομηνια επιστροφης (ΕΕ/ΜΜ/ΗΗ)
    Book *pb;
};
struct Stud {
    int am;           // Αριθμος μητρωου
    char name[30];    // Ονομα σπουδαστη
    Book *pb;  // Δεικτης στην δομη-book
    Stud *ps;  // Δεικτης στην δομη-stud
};
```

Ταξεις (Classes)

Στην C++ η **class** "δημιουργει" ενα νεο τυπο δεδομενων που μπορει να χρησιμοποιηθει να δημιουργησει αντικειμενα (objects) αυτου του τυπου. Οτι ισχυει για τις δομες που οριζονται με struct ισχυουν και για τις δομες που οριζονται με class.

Ορισμος:

```
class ονομα_ταξης {
        δηλωση "ιδιωτικων" δεδομενων και συναρτησεων
    public:
        δηλωση "δημοσιων" δεδομενων και συναρτησεων
} [ονομα αντικειμενου [, ονομα ανρικειμενου [, ...]]];
```

Τα στοιχεια (δεδομενα ή μεταβλητες ή data items) και οι συναρτησεις που δηλωνονται μεσα σε μια **class** ειναι **μελη** (**members**) της class.

Τα μελη μιας ταξης ειναι προσπελασιμα με τρεις τροπους: **private**, **public**, **protected**.

Εξ' ορισμου ολα τα δεδομενα και συναρτησεις σε μια **class** ειναι **private** ("ιδιωτικα"). Αυτο σημαινει οτι μεταβλητες και οι συναρτησεις ειναι προσπελασιμες μονο απο μελη της **class**. Αυτος ειναι ενας απο τους τροπος που η **καψουλοποιηση** (**encapsulation**) επιτυγχανεται δηλ. η προσβαση σε ορισμενες μεταβλητες (δεδομενων) μπορει να ελεγχθει δηλωνοντας αυτες τις μεταβλητες ως private.

Οι μεταβλητες και οι συναρτησεις που οριζονται μετα τη λεξη **public** ("δημοσιες") ειναι προσπελασιμες απο ολες τις αλλες συναρτησεις στο προγραμμα.

Τα μελη μιας ταξης που δηλωνονται με την λεξη **protected** ειναι προσπελασιμα μονο απο μελη της ταξης (τα protected μελη ειναι οπως τα private μελη αλλα διαφερουν οταν κληρονομουνται)

```
class foo {
    int x;
public:
    void set(int w)
    {
        x = w;
    }
    int readx()
    {
        return(x);
    }
};
```

Οι συναρτησεις που υλοποιουνται μεσα στην ταξη οπως η set και η readx ειναι **inline** συναρτησεις.

Η class foo εχει ως μελη: την "ιδιωτικη" μεταβλητη (data item) i, και δυο "δημοσιες" συναρτησεις set και readx.

-- Παραδειγμα

```
#include <iostream.h>
class foo {
    int x;
public:
    void set(int);
    int readx();
    void dsp();
};

void foo :: set(int w)
{
    x = w;
}

int foo :: readx()
{
    return(x);
}

int main(void)
{
    foo A;           // Δηλωση αντικειμενου A τυπου foo
    int z;
    A.set(7);        // Απονομη τιμης στο x
    z = A.readx();   // Διαβασμα της τιμης του x
    cout << z << endl;
    return(0);
}
// p34.cpp //
```

-- Παραδειγμα πινακα απο class.

```
#include <iostream.h>

class foo {
    int x;
public:
    void set(int);
    int readx();
    void dsprev(foo [], int);
    void dsp(foo *, int);
};
void foo::set(int w)
{
    x = w;
}

int foo::readx()
{
    return(x);
}



void foo::dsprev(foo ar[], int n)  // Αναποδη εμφανιση
```

```
{
    for(int i = n - 1 ; i >= 0 ; i--)
        cout << ar[i].x << endl;
}

void foo::dsp(foo *p, int n)    // Με χρηση δεικτη
{
    for(int i = n - 1 ; i >= 0 ; i--) {
        cout << p->x << endl;
        p++;
    }
}

int main()
{
    foo B[2], *p;  // Δηλωση του πινακα
    B[0].set(17);  // Αρχικες τιμες
    B[1].set(13);  //    ...
    B->dsprev(B,2);  // Εμφανιση
    p = B;
    p->dsp(p,2);     // Εμφανιση με δεικτη
    cout << "===" << endl;
    return(0);
}
// p37.cpp //
```

-- Παραδειγμα

```
#include <iostream.h>
#include <string.h>

class Pelatis {
   // Ιδιωτικα στοιχεια
   char   name[30];
   double xreo;
   public:
   // Δημοσια στοιχεια
      long telefon;
      void sname(char *);
      void aname(char *);
      void sxreo(double);
      double axreo();
};

void Pelatis::sname(char *s)  // Σωσε το ονομα του πελατη
{
    strcpy(name, s);
}

void Pelatis::aname(char *s)  // Ανακτηση ονοματος του πελατη
{
    strcpy(s,name);
}


void Pelatis::sxreo(double x)  // Σωσε το χρεος του πελατη
```

```
{
    xreo = x;
}

double Pelatis::axreo()  // Ανακτηση χρεους του πελατη
{
    return(xreo);
}

int main()
{
    Pelatis neos; // Δηλωση αντικειμενου neos (τυπου Pelatis)
    char name[30];
    double y;
    neos.sname("Α. Πιστολάς");
    neos.sxreo(85000);
    neos.telefon = 987456;

    neos.aname(name);
    y = neos.axreo();
    cout << "Ο πελατης " << name << " εχει χρεος " <<
            y << " δρχ." << endl;
    cout << "Τηλεφωνο " << neos.telefon << endl;
    return(0);
}
// p65.cpp //
```

Καψουλοποιηση (Encapsulation)

Καψουλοποιηση ειναι ο μηχανισμος που "δενει" τον κωδικα μαζι με τα δεδομενα και τα κρατα ασφαλη απο εξωτερικη παρεμβαση ή καταχρηση. Η καψουλοποιηση επιτρεπει την δημιουργια αντικειμενου δηλ. ενα αντικειμενο ειναι μια λογικη οντοτητα που καψουλοποιει τα δεδομενα και τον κωδικα που επεξεργαζονται αυτα τα δεδομενα. Σε ενα αντικειμενο "ευαισθητα" δεδομενα και/ή κωδικας μπορουν να δηλωθουν ως private (ιδιωτικα) και δεν ειναι προσπελασιμα σε οτιδηποτε εξω απο το αντικειμενο.
Αλλες περιπτωσεις οπου εχουμε καψουλοποιηση: α) Οι μεταβλητες που δηλωνονται σε ενα block δεν ειναι προσπελασιμες εκτος του block, β) Στην κληρονομηκοτητα, μια παραγομενη ταξη (παιδι) δεν εχει δικαιωμα προσβασης σε private δεδομενα και συναρτησεις της ταξης-βαση (γονεας)

Αντικειμενοστρεφης Προγραμματισμος

Η **`C++ class`** και η σχεση με τον αντικειμενοστρεφη προγραματισμο (**Α.Π**) (**Object oriented progamming - OOP**).
Εαν εχουμε ορισει μια class Pelatis (βλεπε προηγουμενο παραδειγμα)

```
Pelatis neos;
```

Α.Π (OOP)	**C++**
Αντικειμενο (object)	Μια μεταβλητη (instance) της ταξης. Παραδειγμα: **neos**
Μεθοδος (method)	Μια συναρτηση μελος (member function) της ταξης. Παραδειγμα: **Pelatis::aname()**
Ιδιοτητα (attribute)	Ενα στοιχειο (μεταβλητη) της ταξης. Παραδειγμα: **Pelatis.telefon**
Μηνυμα-αποστολη (message-send)	Μια κληση σε συναρτηση-μελος της ταξης. Παραδειγμα: **Pelatis.axreo()**
Μηνυμα-παραλαβη (message-received)	Η τιμη που επιστρεφεται απο μια κληση σε συναρτηση μελος (μηνυμα-αποστολη). Παραδειγμα: **y = Pelatis.axreo()**, το μηνυμα σωζεται στην μεταβλητη y.
Περιβαλλον (interface)	Τα μηνυματα στα οποια ενα αντικειμενο μπορει να ανταποκριθει. Παραδειγμα: το "δημοσιο" μερος της class Pelatis.

Συναρτησεις Δομητων - Αποδομητων

Η **συναρτηση δομητη** (constructor) ειναι μια συναρτηση μελος που εχει το ιδιο ονομα με την **class** ή την **struct**. Σε μια **class** ή μια **struct** ειναι δυνατον να υπαρχουν πολλες συναρτησεις-δομητων αρκει να διαφερουν σε τουλαχιστον μια παραμετρο (βλεπε: Υπερφορτωση συναρτησεων) Η συναρτηση δομητη καλειται να εκτελεσθει οταν ενα αντικειμενο δημιουργειται. Ο δομητης δεν επιστρεφει καμμια τιμη (δεν εχει τυπο και δεν ειναι void)

```
a) Obj();
   Obj(int a);

b) X(int x,int x);

c) CBasi();
   CBasi(int a, char ch);
```

Η **συναρτηση αποδομητη** ή καταστροφεα (desctructor) ειναι συναρτηση μελος που εχει το ιδιο ονομα με την class αλλα επαιται του χαρακτηρα **~** (tilde) Ο αποδομητης δεν επιστρεφει καμμια τιμη (δεν εχει τυπο και δεν ειναι void)

```
a) ~Obj();

b) ~X();

c) ~CBasi();
```

Σε καθε **class** ή **struct** μπορει να υπαρχει μια μονο συναρτηση αποδομητη και δεν φερει καμμια παραμετρο. Ο αποδομητης καλειται στο σημειο που τελειωνει η "ζωη" του αντικειμενου.

-- Παραδειγμα δομητη και αποδομητη.

```
#include <iostream.h>

class Z {
    int x;
    public: Z()  { cout << "Δομητης\n"; };
    ~Z() { cout << "Αποδομητης\n"; };
};

int main()
{
    Z ob;       // Δηλωση αντικειμενου ob. Κληση του δομητη

    return(0);  // Κληση αποδομητη (καταστροφεα)
}
```

Αποτελεσμα

```
Δομητης
Αποδομητης
```

-- Παραδειγμα συναρτησης δομητη που χρησιμοποιειται για να δωσει αρχικη τιμη στο ιδιωτικο μελος x και συναρτησης αποδομητη.

```
#include <iostream.h>

class Obj {
    int x;
    public: Obj(int a);    // Δομητης (constructor)
    ~Obj();                // Αποδομητης (destructor)
};

Obj::Obj(int a)        // Δομητης
{
    x = a;
    cout << "Δομητης x=" << x << endl;
}

Obj::~Obj()   // Αποδομητης
{
    cout << "Αποδομητης x =" << x << endl;
}

int main() {

    Obj o1(1),  // κληση του δομητη για το αντικειμενο o1
        o2(2);  //  ..   ..   ..   ..   ..   ..   ..   o2

    return(0);  // Στο σημειο εξοδου του προγραμματος
                // καλειται ο αποδομητης.
}
// g5.cpp //
```

Αποτελεσμα

```
Δομητης x=1
Δομητης x=2
Αποδομητης x =2
Αποδομητης x =1
```

-- Παραδειγμα δομητη που χρησιμοποιει την γραμμη αρχικοποιησης (initialization line) για να δωση αρχικη τιμη σε μια μεταβλητη μελος.

```
#include <iostream.h>

class Obj {
    int x;
    public: Obj() : x(0) {};   // ιδιο με:  Obj() { x = 0; };
};

int main() {
    Obj o;

    return(0);
}
// g1a.cpp //
```

Πρωτα εκτελειται ο κωδικας μετα το **:** και στην συνεχεια εκτελειται ο κωδικας μεσα στο block του δομητη (εαν περιεχει κωδικα).

-- Ακομη ενα παραδειγμα δομητη που χρησιμοποιει την γραμμη αρχικοποιησης για να δωση αρχικες τιμες σε μεταβλητες μελη.

```
#include <iostream.h>

class Obj {
    int x;
    char ch;
    public:
    Obj(int a,char b) : x(a), ch(b) {
           cout << x << " " << ch;
    };
};

int main() {
    Obj o(3,'R');

    return(0);
}
// g1b.cpp //
```

Αποτελεσμα

```
3 R
```

-- Ακομη ενα παραδειγμα δομητη που χρησιμοποιει την γραμμη αρχικοποιησης για να δωση αρχικη τιμη σε μεταβλητες μελη συνθετης ταξης.

```
#include <iostream.h>

class coord {
    public:
    int x;
    int y;
    coord(int a, int b) { x = a; y = b; };
};

class Obj {
    public:
    coord co;       // αντικειμενο τυπου coord
    char mes[81];
    Obj(int a, int b, char *s);
};

Obj::Obj(int a, int b, char *s) : co(a,b)
{
    strcpy(mes,s);
}

int main() {
    Obj o(5,6,"Dances with wolfs");

    cout << o.co.x << " " << o.co.y << " " << o.mes << endl;

    return(0);
```

```
}
// g1c.cpp //
```

-- Το παρακατω προγραμμα δημιουργει τρια αντικειμενα o1, o2 και o3. Τα αντικειμενα δημιουργουνται με την εξης σειρα o1, o2, o3. Το o3 καταστρεφεται μετα το μπλοκ του if. Τα αντικειμενα o1 και o2 καταστρεφονται λιγο πριν τελειωσει το προγραμμα με την εξης σειρα o2, o1 (αντιθετα με την σειρα που δημιουργηθηκαν).

```
#include <iostream.h>

class Obj {
    int x;    // private
};

int main() {
    Obj o1, o2;

    if (5 > 3) {
       Obj o3;
    }
    // επεξεργασια
    return(0);
}
```

Αντικειμενα ως παραμετροι σε συναρτησεις

Το περασμα αντικειμενων σε συναρτησεις γινεται οπως γινεται σε ολους τους τυπους δεδομενων της C++

Α) Με τιμη (call by value)
Β) Με δεικτη (call by reference με δεικτη)
Γ) Με αναφορα (call by reference με αναφορα)

Α) Με τιμη (call by value)

```
    Obj X;
    :
    func(X);
    :
void func(Obj W)
{
```

```
        ...
    }
```

- Το αντικειμενο X αντιγραφεται στο αντικειμενο W αλλά δεν καλειται ο δομητης (constructor)
- Ο αποδομητης καλειται για το αντικειμενο W στο τελος της συναρτησης func
- Τυχον αλλαγες στο αντικειμενο W μεσα στην συναρτηση func ΔΕΝ επιδρουν στο καλων αντικειμενο X.

Β) Με δεικτη (call by reference με δεικτη)

```
      Obj X;
      :
      func(&X);
      :
    void func(Obj *W)
    {
       ...
    }
```

- Δεν καλειται ο δομητης ουτε ο αποδομητης διοτι στην συναρτηση περναει η διευθυνση του X και οχι το αντικειμενο X.
- Τυχον αλλαγες στο αντικειμενο W μεσα στην συναρτηση func επιδρουν στο καλων αντικειμενο X.

Γ) Με αναφορα (call by reference με αναφορα)

```
      Obj X;
      :
      func(X);
      :

    void func(Obj& W)
    {
       ...
    }
```

- Δεν καλειται ο δομητης ουτε ο αποδομητης διοτι στην συναρτηση χρησιμοποιειται το συνονυμο αντικειμενο (reference μεταβλητη) W του X οχι το αντικειμενο X.
- Τυχον αλλαγες στο αντικειμενο W μεσα στην συναρτηση func επιδρουν στο καλων αντικειμενο X.

Φιλες συναρτησεις (`friend` functions)

Ειναι δυνατον σε συναρτησεις μη-μελη να εχουν προσβαση σε "ιδιωτικα" (**`private`**) μελη μιας ταξης οταν αυτες δηλωνονται ως φιλες συναρτησεις. Μια φιλη συναρτηση εχει προσβαση σε ολα τα "ιδιωτικα" και "δημοσια" μελη μιας ταξης. Το πρωτοτυπο της φιλης συναρτησης δηλωνεται μεσα στην ταξη με προθεμα την λεξη **`friend`**.

```
#include <iostream.h>

class Z {
    int x;
    public:
    Z(int a) { x = a; };
    friend int showx(Z ob);     // Δηλωση φιλη συναρτηση
};

int showx(Z obj)      // Υλοποιηση φιλης συναρτηση
```

```
{
    cout << obj.x;   // προσβαση σε private μεταβλητη μελος x
του obj
}

int main()
{
    Z ob(4);      // Δηλωση αντικειμενου ο (x = 4)

    showx(ob);    // Κληση της φιλης συναρτησης

    return(0);
}
// g39.cpp //
```

Αποτελεσμα

```
 4
```

Στατικα μελη (`static` members)

Ως **`static`** μπορουν να δηλωθουν αντικειμενα και συναρτησεις.

α) Δηλωση στατικων αντικειμενων

```
static int a;
static Obj b;
```

β) Δηλωση στατικης συναρτησης

```
static int getx();
```

Ως στατικες (**`static`**) επισης μπορουν να δηλωθουν μεταβλητες (static data members) και συναρτησεις μελη (static member functions) σε μια **`class`**.

Στατικες μεταβλητες μελη

Οταν μια μεταβλητη δηλωνεται ως **static** σε μια ταξη (static data member) ο μεταφραστης κρατα μονο μια "κόπια" αυτης της μεταβλητης την οποια μοιραζονται ολα τα αντικειμενα της ταξης. Μονο μια "κοπια" της στατικης μεταβλητης υπαρχει ανεξαρτητως απο το ποσα αντικειμενα αυτης της ταξης δημιουργουνται. Κατ' αυτον τον τροπο ολα τα αντικειμενα αυτης της ταξης χρησιμοποιουν την ιδια μεταβλητη.

```
class Obj {
    :
    static int x;
    :
};

int Obj::x;
```

Η στατικη μεταβλητη υπαρχει πριν απο την δημιουργια του πρωτου αντικειμενου αυτης της ταξης και λαμβανει τιμη αρχικη τιμη μηδεν.

-- Παραδειγμα, Στατικης μεταβλητης μελος ταξης

```
#include <iostream.h>

class Obj {
    static int x;  // Στατικη μεταβλητη μελος
    public:
    Obj() { };
    void setx(int a) { x = a; };
    int getx() { return(x); };
};

int Obj::x;

int main()
{
    Obj o,b;

    cout << o.getx() << endl;    // 0

    o.setx(5);

    cout << o.getx() << endl;    // 5
    cout << b.getx() << endl;    // 5

    return(0);
}
// g15.cpp //
```

Αποτελεσμα

```
   0
   5
   5
```

Στατικες συναρτησεις μελη

Οι στατικες συναρτησεις μιας ταξης (static member functions) εχουν προσβαση μονο σε στατικες μεταβλητες αυτης της ταξης, και δεν εχουν **this** δεικτη. Μια στατικη

συναρτηση δεν μπορει να εχει και μη-στατικη (non-static) version της ιδιας συναρτησης. Εχουν περιορισμενη χρηση και χρησιμοπουνται για την αρχικοποιηση στατικων μεταβλητων πριν απο οποιαδηποτε δημιουργια αντικειμενων.

-- Παραδειγμα, Στατικης συναρτησης μελος ταξης.

```
#include <iostream.h>

class Obj {
    static int x;
    public:
    Obj() { };
    static void setx(int a) { x = a; };
    static int getx();
};

int Obj::x;

int Obj::getx()
{
   return(x);
}

int main()
{
    Obj A, B;   // Το Α και Β μοιραζονται την μεταβλητη x

    Obj::setx(3);// Επειδη η setx ειναι static δεν χρειαζεται
                 // αντικειμενο να εντελεσθει
    cout << "Του Α το x = " << A.Obj::getx() << endl;
    cout << "Του Β το x = " << B.getx() << endl;

    A.setx(5);
    cout << "Του Α και Β το x = " << Obj::getx() << endl;
    return(0);
} // g85.cpp //
```

Αποτελεσμα

```
Του Α το x = 3
Του Β το x = 3
Του Α και Β το x = 5
```

`const` και `volatile` συναρτησεις μελη

Οι συναρτησεις μελη μπορουν να δηλωθουν ως **`const`** ή **`volatile`** ή **`const volatile`**.

- **`volatile`** αντικειμενα μπορουν να εκτελεσουν μονο **`volatile`** συναρτησεις μελη.
- **`volatile`** συναρτησεις μελη μπορουν να εκτελεσθουν απο **`volatile`** και μη-volatile αντικειμενα.

- **`const`** αντικειμενα μπορουν να εκτελεσουν μονο **`const`** συναρτησεις μελη.
- **`const`** συναρτησεις μελη μπορουν να εκτελεσθουν απο **`const`** και μη-const αντικειμενα.

Συνοψιζοντας εχουμε:

```
                  OK    OK    Err     OK    OK    Err
                  ---   ---   ---     ---   ---   ---
Συναρτηση-μελος:   c     c    μ-c      v     v    μ-v
Αντικειμενο    :   c    μ-c    c       v    μ-v    v

οπου  c = const,  v = volatile,
     μ-c = μη-const, μ-v = μη-volatile
```

Οι παραπανω κανονες συνδιαζονται οταν αντικειμενα και συναρτησεις μελη δηλωνονται ως **const volatile**.

```
                  OK    OK    OK    OK    Err
                  ---   ---   ---   ---   ---
Συναρτηση-μελος: c-v   c-v   c-v   c-v   ocv
Αντικειμενο ...: c-v    v     c    ocv   c-v

οπου  c-v = const volatile,
      ocv = ουτε const, ουτε volatile, ουτε const volatile
```

Οι **const** συναρτησεις-μελη δεν μπορουν να μεταβαλουν μεταβλητες μελη.

```
-- Παραδειγμα δηλωσης συναρτησεων-μελη ως const ή volatile:
class Obj {
    public:
    int x;
    Obj(int a) { x = a; };
    void setxc(int a)  const;
    void setxv(int a)  volatile;
    void setxcv(int a) const volatile;
};
```

-- Παραδειγμα, **const** συναρτηση-μελος (Η **const** συναρτηση μελος getx δεν μπορει να αλλαξει την μεταβλητη x μελος της **class**)

```
#include <iostream.h>

class Obj {
    public:
    int x;
    void setx(int a) { x = a; };
    int getx() const;
;

int Obj::getx() const
{
  // x = 33;       // ΛΑΘΟΣ // Δεν μπορει να αλλαξει τιμη σε
  // Obj::setx(55);// ΛΑΘΟΣ \\ μεταβλητες-μελη λογω του const

    return x;
```

```
}

int main()
{
    Obj A;

    A.setx(6);

    cout << A.getx() << endl;

    return(0);
}
// g20.cpp //
```

Δεικτης this

Οταν μια συναρτηση μελος καλειται, αυτοματα περναει και μια αφανης παραμετρος που ειναι ενας δεικτης στο αντικειμενο που δημιουργησε την κληση. Ο δεικτης αυτος ονομαζεται **this** pointer. Ο δεικτης **this** εφαρμοζεται μονο σε συναρτησεις μελη. Οι στατικες (**static**) και οι φιλες (**friend**) συναρτησεις δεν εχουν τον δεικτη **this**.

-- Παραδειγμα. Το προγραμμα υπολογιζει το τετραγωνο ενος αντικειμενου (μεταβλητη x)

```
#include <iostream.h>

class Obj {
    int x;
    public:
    Obj(int a) { x = a; };
    int getx2();    // συναρτηση μελος
};

int Obj::getx2() {
    int b;
    b = (this->x)*(this->x);   // ιδιο με: b = x*x;
    return(b);
}

int main()
{
    Obj A(3);       // Δηλωση αντικειμενου A.  x = 3;
    int k;
    k = A.getx2();// Το A καλει την getx2 συναρτηση-μελος και
                  // περνά τον δεικτη this που δειχνει στο A.
                  // Υπολογισμος του x*x
    cout << k << endl;
    return(0);
}
// g37.cpp //
```

Αποτελεσμα

```
  9
```

-- Παραδειγμα. Επιστροφη αντικειμενου απο συναρτηση με δεικτη **this**. Το προγραμμα προσθετει εναν εκεραιο σε ενα αντικειμενο (μεταβλητη x). Το αντικειμενο ***this** που επιστρεφει η συναρτηση RetObj ειναι το αντικειμενο που δημιουργησε την κληση.

```
#include <iostream.h>

class Obj {
    int x;
    public:
    Obj(int a) { x = a; };
    Obj RetObj(int a);
    int getx() { return(x); };
};

Obj Obj::RetObj(int a) {
    this->x = this->x + a;   // ιδιο με:  x = x + a;
    return(*this);
}

int main()
{
    Obj A(3);         // Δηλωση αντικειμενου A.  x = 3;

    A.RetObj(10);   // Το A δημιουργει την κληση της RetObj
                    // Υπολογισμος x + 10
    cout << A.getx() << endl;
           // Εμφανιση νεας τιμης του x στο A.

    return(0);
}
// g38.cpp //
```

Αποτελεσμα

```
13
```

Δεικτες σε μελη ταξης (pointers to `class` members)

Ενας δεικτης σε μελος ταξης ειναι ενας ειδικος δεικτης που δειχνει σε ενα μελος ταξης γενικα, και οχι σε ενα μελος ενος αντικειμενου αυτης της ταξης. Οι δεικτες σε μελη ταξης μπορουν να δειξουν μονο σε μελη μιας ορισμενης ταξης. Οι δεικτες σε μελη ταξης δηλωνονται με τον τελεστη **::***

```
α) int Obj::*data; // Ο δεικτης data δυναται να δειξει σε int
                   // μεταβλητη μελος της ταξης Obj.

b) void (STR::*fn)(); // Ο δεικτης fn δυναται να δειξει σε
                      // void συναρτηση μελος χωρις
                      // παραμετρους της ταξης STR.

γ) long (X::*pF[2])(char); // Πινακας pF απο δυο δεικτες-σε-
       // μελη.  Οι δεικτες  δυνανται να δειξουν σε long
       // συναρτησεις μελη με μια char παραμετρο της ταξης X.
```

Για να εχουμε προσβαση σε μελη οταν χρησιμοποιουνται δεικτες-σε-μελη πρεπει κανουμε χρηση των τελεστων **.*** ή **->*** Ο τελεστης **.*** χρησιμοποιειται οταν χρησιμοποιουμε αντικειμενο. Ο τελεστης **->*** χρησιμοποιειται οταν χρησιμοποιουμε δεικτη σε αντικειμενο.

-- Παραδειγμα δεικτη σε μεταβλητη μελος-ταξης

```
#include <iostream.h>

class Obj {
    public:
    int x;
};

int main()
{
    int Obj::*pti;     // Δηλωση δεικτη-σε-μελος

    pti = &Obj::x;     // Ο δεικτης δειχνει στο x

    Obj A;             // Δηλωση αντικειμενου

    A.x = 5;           // x = 5

    cout << A.*pti;// Προσβαση στο x μεσω του δεικτη-σε-μελος

    Obj *pA;           // Δηλωση δεικτη pA στο αντικειμενο A

    pA = &A;           // Ο δεικτης pA δειχνει στο A

    cout << pA->*pti; // Προσβαση στο x μεσω του
                       // δεικτη-σε-μελος
}
// g32.cpp //
```

Αποτελεσμα

```
 5
 5
```

-- Παραδειγμα με δεικτες σε μεταβλητες και συναρτησεις μελη ταξης

```
#include <iostream.h>

class Pers {
    public:
    int nr;
    void set(int a) { nr = a; };
    int get() { return(nr); };
};

int main()
{
    // Δηλωσεις δεικτων σε μελη της ταξης Pers
    int   Pers::*pnr;
    void (Pers::*ps)(int);
    int  (Pers::*pg)();
```

```
    // Οι δεικτες δειχνουν σε μελη
    pnr = &Pers::nr;
    ps  = &Pers::set;
    pg  = &Pers::get;

    Pers   P1,   P2; // Αντικειμενα P1 και P2
    Pers *pP1, *pP2; // pP1 και pP2 δεικτες σε αντικειμ Pers

    (P1.*ps)(3);   // ιδιο με  P1.set(3)
    (P2.*ps)(9);   //          P2.set(9)

    // Εμφανιση κανοντας χρηση του δεικτες σε μελη
    cout << "Τιμες με δεικτη σε μελη\n";
    cout << P1.*pnr  << "  " << P2.*pnr  << endl;
    cout << (P1.*pg)() << "  " << (P2.*pg)() << endl;

    pP1 = &P1;   // Ο δεικτης pP1 δειχνει στο P1
    pP2 = &P2;   // Ο δεικτης pP2 δειχνει στο P2

    (pP1->*ps)(23);  // ιδιο με pP1->set(23)
    (pP2->*ps)(29);  //         pP2->set(29)

    // Εμφανιση κανοντας χρηση του δεικτη σε αντικειμενο και
    // του δεικτη σε μελη
    cout << "Τιμες με δεικτη-σε-αντικ και δεικτη σε μελη\n";
    cout <<  pP1->*pnr   << "  " <<  pP2->*pnr  << endl;
    cout << (pP1->*pg)() << "  " << (pP2->*pg)() << endl;

    return(0);
} // g30.cpp //
```

Αποτελεσμα

```
Τιμες με δεικτη σε μελη
3  9
3  9
Τιμες με δεικτη-σε-αντικ και δεικτη σε μελη
23  29
23  29
```

-- Παραδειγμα, πινακα απο δεικτες σε συναρτησεις μελη.

```
#include <iostream.h>

class Pers {
    public:
    int nr;
    void set(int a) { nr = a; };
    int get() { return(nr); };
    int getd() { return(nr+nr); };  // Διπλά
};

int main()
{
    // Δηλωση πινακα δεικτων (δυο κελια) σε int συναρτησεις
    // (χωρις παραμετρους) που ειναι μελη της ταξης Pers
    int  (Pers::*pg[2])();

    // Οι δεικτες δειχνουν σε μελη
```

```
    pg[0] = &Pers::get;
    pg[1] = &Pers::getd;

    Pers   P;  // Αντικειμενο P
    Pers *pP;  // pP δεικτης σε αντικειμενα Pers

    P.set(3);

    // Εμφανιση κανοντας χρηση του πινακα απο δεικτες σε
    // συναρτησεις μελη
    cout << "Τιμες με δεικτη σε μελη\n";
    cout << (P.*pg[0])() << " επι 2 = " << (P.*pg[1])()
         << endl;

    pP = &P;   // Ο δεικτης pP δειχνει στο P

    pP->set(23);

    cout << "Τιμες με δεικτη-σε-αντικ και δεικτη σε μελη\n";
    cout << (pP->*pg[0])() << " επι 2 = " << (pP->*pg[1])()
         << endl;

    return(0);
}
// g31.cpp //
```

Αποτελεσμα

```
Τιμες με δεικτη σε μελη
3 επι 2 = 6
Τιμες με δεικτη-σε-αντικ και δεικτη σε μελη
23 επι 2 = 46
```

Υπερφορτωση Τελεστων

Εκτος απο συναρτησεις μπορουμε να υπερφορτωσουμε και τελεστες (operator overloading) ωστε να εκτελουν ειδικες λειτουργιες σε αντικειμενα ταξεων που εχουμε δημιουργησει. Η υπερφορτωση τελεστων γινεται με συναρτησεις και δηλωνονται ως εξης:

a) ***τυπος*** **operator** ***τελεστης*****();**
b) ***τυπος*** **operator** ***τελεστης*****(*****παραμετροι*****);**

Παραδειγματα

```
a) Obj operator++();
b) int operator==(Obj a,Obj b);
```

-- Παραδειγμα υπερφορτωσης τελεστη **+** με συναρτηση μελος ωστε να ειναι δυνατον να προσθετουμε σε ενα αντικειμενο Α ταξης Obj εναν ακεραιο
(Obj B = A + ακεραιο;)

```
#include <iostream.h>
class Obj {
    int x;
    public:
    Obj(int a) { x = a; };
    int getx { return(x); };
    Obj operator+(int w);
};

Obj Obj::operator+(int w)  // Υπερφωρτωση τελεστη
{
    Obj tmp(0);
    tmp.x = x + w;      // ιδιο με: tmp.x = this->x + w;
    return(tmp);
}

int main()
{
    Obj A(6), B(0);

    B = A + 7;     // ιδιο με: B = A.operator+(7);
                   // Το αντικειμενο A δημιουργει την κληση
    cout << "To x του B = " << B.getx() << endl;

    return(0);
}
```

Αποτελεσμα

```
To x του B = 13
```

Τελεστες που μπορουν να υπερφορτωθουν

= (μόνο με συναρτηση-μελος)

+ - * /

+= -= *= /=

== > < >= <=

`++ --` Προθεμα πχ. `++o --o`

```
clastype operator++();  // Πρωτοτυπο
clastype operator--();  // Πρωτοτυπο
```

Επιθεμα πχ. `o++ o--`

```
clastype operator++(int);  // Πρωτοτυπο
clastype operator--(int);  // Πρωτοτυπο
```

`new delete`

`[]` (μόνο με συναρτηση-μελος)

```
τυπος τυπος::operator[](int x) // Πρωτοτυπο
{
   :
}
```

`()` (μόνο με συναρτηση-μελος)

`->` (μόνο με συναρτηση-μελος)

`,` (κομμα)

`<<` insertion operator (παρεμβαλει χαρακτηρες σε μια ροη)

Επικεφαλιδα συναρτησης

```
friend ostream &operator<<(ostream& stream,Obj o)
{
     :
}
```

`>>` extraction operator (εξαγει χαρακτηρες απο μια ροη)

Επικεφαλιδα συναρτησης

```
friend istream &operator>>(istream& stream,Obj o)
{
    :
}
```

Οι τελεστες `=, [], (), ->` υπερφορτωνονται μονο με συναρτησεις-μελη.

Οι τελεστες `. :: .* ?` δεν μπορουν να υπερφορτωθουν.

Οι συναρτησεις υπερφορτωσης-τελεστων μπορουν να ειναι:
α) συναρτησεις μελη (member functions)
β) φιλες συναρτησεις (friend functions)
γ) συναρτησεις μη-μελη (non-member functions)(σε αυτην την περιπτωση οι συναρτησεις εχουν προσβαση μονο σε δημοσια μελη της class)
- Ο τροπος β) παραβιαζει εν μερει την αρχη της καψουλοποιησης και ο τροπος γ) παραβιαζει κατάφορα την αρχη της καψουλοποιησης.

`--` Παραδειγμα, Υπερφορτησης τελεστων με συναρτηση μελος, με φιλη συναρτηση και μη-μελος συναρτηση

```
#include <iostream.h>

class Obj {     // Ταξη
    int x;
    public:
    int z;

    Obj(int a,int b);   // Δομητης

    void Obj::get(int& x, int& b);

    Obj operator-(Obj &o);

    friend Obj operator*(Obj &o1, Obj &o2);
    // friend Obj operator^(Obj &o1, Obj &o2);  // Αντι του +
};

Obj::Obj(int a, int b)  // Δομητης
{
    x = a;
    z = b;
}

void Obj::get(int& x, int& b)
{
   x = Obj::x; // Χρειαζεται ο τελεστης θεας για να
               // ξεχωριζει η παραμετρος x απο το x της class
   b = z;
}

Obj Obj::operator-(Obj& o)        // Συναρτηση μελος
{
    Obj tmp(0,0);
    tmp.x = x - o.x;
    tmp.z = z - o.z;
    return(tmp);
}

Obj operator*(Obj &o1, Obj &o2)   // Φίλη συναρτηση
{
    Obj tmp(0,0);
    tmp.x = o1.x * o2.x;
    tmp.z = o1.z * o2.z;
    return(tmp);
}

int operator>(Obj& o1,Obj& o2)    // Μη-μελος συναρτηση
{

    return(o1.z > o2.z ? 1 : 0);  // Προσβαση μονο σε public
}                                 // μελη

int main()
{
    Obj A(2,3),B(9,8),C(0,0);
```

```
    int x,y;

    C = B - A;   // Κληση της συναρτησης operator-
    C.get(x,y);
    cout << "C.x = " << x << "  C.z = " << y << endl;

    C = A * B;   // Κληση της συναρτησης operator*
    C.get(x,y);
    cout << "C.x = " << x << "  C.z = " << y << endl;

    if (B > A) // Κληση της συναρτησης operator>
       cout << "Το B.z μεγαλυτερο του A.z" << endl;
    else
       cout << "Το A.z μεγαλυτερο του B.z" << endl;

    return(0);
}
// g9.cpp //
```

Παρατηρησεις

Η εντολη `C=B - A;` μπορει να γραφει και ως `C = B.operator-(A);`
Η εντολη `C=B * A;` μπορει να γραφει και ως `C = operator*(A,B);`
Η συγκριση `(B > A)` μπορει να γραφει και ως `(operator>(B,A))`

Αντιγραφεας Δομητης (copy constructor)

Οταν ενα αντικειμενο χρησιμοποιειται για να δωσει τιμη σε ενα αλλο αντικειμενο τοτε εκτελειται bit προς bit αντιγραφη.

```
    Obj A, B;
    :
    B = A;  // Bit προς bit αντιγραφη του A στο B.
            // Το B ειναι ακριβες αντιγραφο του A.
```

Αν ο προγραμματιστης θελει να κανει μερικη αντιγραφη απο αντικειμενο σε αντικειμενο τοτε δεν ειναι επιθυμητη η bit προς bit αντιγραφη.

Η bit προς bit αντιγραφη δεν ειναι επιθυμητη επισης οταν τα αντικειμενα δεσμευουν μνημη κατα την δημιουργια τους. Αν υποθεσουμε οτι το αντικειμενο A εχει δεσμευσει μνημη τοτε με την bit προς bit αντιγραφη του A στο B, το αντικειμενο B χρησιμοποιει την ιδια μνημη του χρησιμοποιει το A. Ενω θα επρεπε το B να χρησιμοποιει την δικη του μνημη. Επι πλεον εαν καταστραφεί το A τοτε ελευθερωνει την μνημη με συνεπεια να καταστρεφει και το B.

Το ιδιο προβλημα δημιουργειται σε ακομη δυο περιπτωσεις:

α) Οταν ενα αντικειμενο A περνα ως παραμετρο σε συναρτηση τοτε δημιουργειται ενα αντιγραφο του A το B. Το αντικειμενο B καταστεφεται μολις τελειωση η εκτελεση της συναρτησης func.

```
    Obj A;
    :
    func(A);
```

```
    :

    void func(Obj B)
    {
        :
    }
```

β) Οταν μια συναρτηση επιστρεφει αντικειμενο. Η συναρτηση func επιστρεφει ενα αντικειμενο τυπου Obj. Το temp αντιγραφεται στο Α και μετα καταστρεφεται.

```
    Obj A;
    :
    A = func();
    :

    Obj func()
    {
        Obj temp;
        :
        return(temp)'
    }
---
```

Το προηγουμενο προβλημα λυνεται με τον αντιγραφεα-δομητη (copy constructor) και εχει την μορφη:

```
    ονομα_ταξης(const ονομα_ταξης &ob) {
        σωμα δομητη
    }
```

Οταν υπαρχει συναρτηση αντιγραφεας-δομητης ο μεταφραστης (compiler) χρησιμοποιει τον αντιγραφεα δομητη (αντι της bit προς bit αντιγραφη) οταν χρειαζεται ενα αντικειμενο να αρχικοποιησει ενα αλλο.

-- Απλο παραδειγμα χρησης συναρτησης αντιγραφεα-δομητη.

```
#include <iostream.h>

class Obj {
    int x;
    public:
    Obj(int a); // Δομητης (constructor)
    ~Obj();     // Αποδομητης (destructor)

    Obj(const Obj& o);  // Αντιγραφεας-Δομητης
                        // (copy contructor)
};

Obj::Obj(int a)        // Δομητης
{
    x = a;
    cout << "Δομητης=" << x << endl;
}

Obj::Obj(const Obj& o)  // Υλοποιηση του Αντιγραφεα-Δομητη
{
    x = o.x;
```

```
    cout << "Αντιγραφεας-Δομητης=" << x << endl;
}

Obj::~Obj()   // Καταστροφεας
{
    cout << "Καταστροφεας=" << x << endl;
}

void func(Obj ob)  // Συναρτηση μη-μελος
{
    //
}

int main()
{
    Obj o1(5);

    cout << "\n----- Πριν την κληση -----\n";

    func(o1);  // κληση με τιμη (call by value)

    cout << "\n----- Μετα την κληση -----\n";

    return(0);
}
// g88.cpp //
```

Αποτελεσμα	Δραση
Δομητης=5	- Obj o1(5)
----- Πριν την κληση -----	
Αντιγραφεας-Δομητης=5	- Το o1 αντιγραφεται στο ob
Καταστροφεας=5	- Το ob καταστρεφεται
----- Μετα την κληση -----	
Καταστροφεας=5	- Το o1 καταστρεφεται

-- Παραδειγμα, Αντιγραφεα-δομητη και υπερφορτωση τελεστη = Τα αντικειμενα κανουν δυναμικη χρηση μνημης.

```
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>

class Obj {
    char *mes;
    public:
    Obj(char *s);        // Δομητης (constructor)
    Obj(const Obj& o);   // Αντιγραφεας-Δομητης
                         // (copy contructor)
    ~Obj();              // Καταστροφεας (destructor)

    char *get();
```

```
    Obj operator=(Obj &o); // Υπερφορτωση τελεστη =
};

char* Obj::get()
{
    return(mes);
}

Obj Obj::operator=(Obj& o)  // Υπερφορτωση τελεστη =
{
    cout << "\n-- Μεσα στην συναρτηση --" << endl;
    cout << "Καλών αντικειμενο this->mes=" << this->mes
         << endl;
    cout << "Αντικειμενο ως παραμετρο o.mes=" << o.mes
         << "\n";
    delete mes;
    mes = new char [strlen(o.mes)+1];
    strcpy(mes,o.mes);
    return(*this);
}

Obj::Obj(char *s)       // Δομητης
{
    int a;
    a = strlen(s);;
    mes = new char [a+1]; // +1 για τον τερματικο χαρακτ '\0'
    strcpy(mes,s);
    cout << "Δομητης=" << mes << endl;
}

Obj::Obj(const Obj& o)  // Αντιγραφεας-Δομητης
{
    int a;
    a = strlen(o.mes);
    mes = new char [a+1];
    for(int i = 0; i <= a; i++) mes[i] = o.mes[i];
    cout << "Αντιγραφεας-Δομητης=" << mes << endl;
}

Obj::~Obj()   // Καταστροφεας
{
    cout << "Καταστροφεας=" << mes << endl;
    delete mes;
}

int main()
{
    Obj o1("CROSS ROAD"), o2("BLUES"); // Κληση δομητη
                                       // για το o1 και o2

    o2 = o1;  // Κληση της συναρτησης-υπερφορτωση τελεστη =

    cout << "\n------ Μετα την κληση \n";
```

```
    cout << "o2 message=" << o2.get() << "\n\n";

    return(0);
}
// g4.cpp
```

Αποτελεσμα

```
Δομητης=CROSS ROAD
Δομητης=BLUES

-- Μεσα στην συναρτηση --
Καλών αντικειμενο this->mes=BLUES
Αντικειμενο ως παραμετρο o.mes=CROSS ROAD
Αντιγραφεας-Δομητης=CROSS ROAD
Καταστροφεας=CROSS ROAD

------ Μετα την κληση
o2 = CROSS ROAD

Καταστροφεας=CROSS ROAD
Καταστροφεας=CROSS ROAD
```

Προστατευομενα (`protected`) μελη σε `class`

Οταν ενα μελος σε μια **`class`** δηλωνεται ως **`protected`** σημαινει οτι αυτο το μελος ειναι προσπελασημο μονο απο συναρτησεις μελη. Ειναι οπως ενα **`private`** μελος αλλα διαφερει οταν κληρονομειται.

Κληρονομηκοτητα (Inheretance)

Η κληρονομηκοτητα ειναι ενα κυριο χαρακτηριστικο στο αντικειμενοστρεφη προγραμματισμο. Η κληρονομηκοτητα επιτρεπει την δημιουργια ιεραρχιας απο ταξεις (classes) αρχιζοντας απο την πιο γενικη προχωροντας προς την πιο ειδικη. Για να εχουμε κληρονομηκοτητα πρεπει πρωτα να ορισουμε μια ταξη βαση (base class) η οποια οριζει τις γενικες ιδιοτητες για ολα τα αντικειμενα τα οποια θα παραχθουν απο την βαση. (Η ταξη βαση ονομαζεται και ταξη "γονέας" (parent class))

Οι ταξεις που παραγονται απο την βαση ονομαζονται παραγομενες ταξεις (derived classes). (Η παραγομενη ταξη ονομαζεται και ταξη "παιδι") Μια παραγομενη class περιλαμβανει ολα τα χαρακτηριστικα της γενικης ταξης-βαση και στην συνεχεια προσθετει ιδιοτητες που ειναι ειδικες στην παραγομενη ταξη.

```
class παραγομενη_ταξη : <τυπος_προσβασης> βαση {
   // δηλωσεις μελων παραγομενης
};
```

Ο ***τυπος_προσβασης*** ειναι ο τροπος με τον οποιο η παραγομενη κληρονομει δικαιωματα απο την βαση και μπορει να ειναι:

public Ολα τα **public** μελη του "γονεα" θα ειναι **public** και στο "παιδι" που το κληρονομει. Ολα τα **protected** μελη του "γονεα" θα ειναι **protected** στο "παιδι". Τα **private** μελη του "γονεα" παραμενουν **private** στον "γονεα" και δεν ειναι προσπελασημα στο "παιδι"

private Ολα τα **public** και **protected** μελη του "γονεα" θα γινουν **private** μελη στο "παιδι".

protected Ολα τα **public** και **protected** μελη του "γονεα" θα γινουν **protected** μελη στο "παιδι".

- Εαν λειπει ο **τυπος_*προσβασης*** τοτε εξ' ορισμου ειναι **private**.

Παραδειγματα ιεραρχειας ταξεων

```
α)          _Ζωο_
           /  |  \
      Γατα  Αρνι Αγελαδα

β)   Οικιακη_Συσκευη
           / \
     Κουζινα  Ψυγειο

γ)   Ζωο_Ξηρας   Ζωο_Υδροβιο
              \ /
          Ζωο_Αμφιβιο
          (Βατραχος)
```

-- Παραδειγμα, Κληρονομικοτητας

```
          Οχημα
           / \
      Κουρσα   Φορτηγο
```

```
#include <iostream.h>

class Mac {     // Ταξη βασης, Οχημα
    int le;     // μηκος σε μ
    int hp;     // ισχυς σε αλογα
    public:
    set_le(int l) { le = l; };
    set_hp(int p) { hp = p; };

    int get_le() { return le; };
    int get_hp() { return hp; };
```

```
};

class Car : public Mac {  // Παραγομενη ταξη, επιβατικο
    int seats;  // καθησματα
    public:
    set_seats(int s) { seats = s; };
    int get_seats() { return seats; };
};

class Track : public Mac {  // Παραγομενη ταξη, φορτηγο
    int load;   // φορτιο σε τονους
    public:
    set_load(int lo) { load = lo; };
    int get_load() { return load; };
};

int main()
{
    Car   A;  // επιβατικο A

    A.set_le(6);
    A.set_hp(75);
    A.set_seats(8);

    cout << "Το αυτοκινητο εχει " << A.get_hp()
         << " αλογα ισχυ\n";

    Track F;  // φορτηγο F

    F.set_le(6);
    F.set_hp(75);
    F.set_load(14);

    cout << "Το φορτηγο εχει " << F.get_load()
         << " τονους φορτιο\n";

    return(0);
}
// g11.cpp //
```

-- Παραδειγμα, Κληρονομικοτητα Πολλαπλης Βασης. Η παραγομενη ταξης κληρονομει απο πολλες βασεις.

```
       B1   B2
         \ /
          Π
```

```
#include <iostream.h>

class Basi1 {
        int x;
    public:
        setx(int a) { x = a; };
        int getx() { return(x); };
};
```

```
class Basi2 {
    int y;
    public:
    sety(int b) { y = b; };
    int gety() { return(y); };
};

class Paragomeni : public Basi1, public Basi2 {
    public:
    getxy(int &a,int &b) { a = getx(); b = gety(); };
            // Δικαιωμα απο κληρονομια για προσβαση σε public
            // συναρτησεις μελη της Basi1 και Basi2
};

int main()
{
    Paragomeni ob;

    ob.setx(4);  // Δικαιωμα απο κληρονομια για προσβαση σε
    ob.sety(15); // public συναρτησεις της Basi1 και Basi2

    int a,b;

    ob.getxy(a,b); // απο την "παραγομενη"

    cout << a << "  " << b << endl;

    return(0);
}
// g40.cpp //
```

Αποτελεσμα

```
4  15
```

-- Παραδειγμα, Κληρονομικοτητα Πολλαπλης Βασης. Η παραγομενη ταξη κληρονομει απο πολλες βασεις.

```
     B1   B2
       \ /
        Π
```

```
#include <iostream.h>

class Basi1 {
    protected:
        int x;
    public:
        int getx() { return(x); };
};

class Basi2 {
    protected:
        int y;
    public:
        int gety() { return(y); };
};
```

```
class Paragomeni : public Basi1, public Basi2 {
    public:
    setxy(int a,int b) { x = a; y = b; };
         // Δικαιωμα απο κληρονομια για προσβαση σε protected
         // μεταβλητες μελη x, y της Basi1 και Basi2
};

int main(void)
{
    Paragomeni ob;

    ob.setxy(4,15);

    int a,b;

    a = ob.getx();    // απο κληρονομια της Basi1
    b = ob.gety();    // απο κληρονομια της Basi2

    cout << a << " " << b << endl;
    return(0);
} // g41.cpp //
```

Αποτελεσμα

```
4  15
```

-- Παραδειγμα, Πολλαπλης Κληρονομικοτητας. Η παραγομενη ταξη ειναι βαση για αλλη παραγομενη.

```
        A
        |
        B
        |
        Γ
```

```
#include <iostream.h>

class Basi {
    int x;
    public:
    void setx(int a) { x = a; };
    int getx() { return(x); };
};

class Paragomeni1 : public Basi {
    int y;
    public:
    sety(int a) { y = a * getx(); };
    int gety() { return(y); };
};

class Paragomeni2 : public Paragomeni1 {
    int z;
    public:
    setz(int a) { z = a - gety(); };
    int getz() { return(z); };
};
```

```
int main()
{
    Paragomeni2 ob;

    ob.setx(4);
    ob.sety(15);
    ob.setz(96);

    int a,b,c;

    a = ob.getx();
    b = ob.gety();
    c = ob.getz();

    cout << a << "  " << b << "  " << c << endl;

    return(0);
}
// g42.cpp //
```

Δομητες - Αποδομητες και Κληρονομικοτητα

Ειναι δυνατον μια "βαση" ταξη να εχει δομητη, οπως και μια "παραγομενη" ταξη να εχει δομητη, ή και οι δυο (βαση και παραγομενη) να εχουν δομητη.

Οταν ενα αντικειμενο απο μια παραγομενη ταξη δημιουργειται τοτε θα εκτελεσθει πρωτα ο δομητης της βασης και μετα ο δομητης της παραγομενης.

Οταν ενα παραγομενο αντικειμενο "τελειωνει" τοτε εκτελειται ο αποδομητης της παραγομενης και μετα ο αποδομητης της βασης.

Οταν εχουμε πολλαπλη κληρονομικοτητα (η παραγομενη ειναι βαση για αλλη παραγομενη τοτε οι δομητες καλουνται με την σειρα που "παραγονται", οι αποδομητες με αντιθετη σειρα.

Οταν εχουμε κληρονομικοτητα πολλαπλων βασων τοτε η παραγομενη εχει την εξης μορφη:

```
class παραγομενη : <τυπος_προσβασης> βαση1,
                   <τυπος_προσβασης> βαση2,
                                .. ..
                   <τυπος_προσβασης> βασηN
{
    // σωμα παραγομενης ταξης
}
```

Η παραγομενη κληρονομει απο *βαση1* εως και *βασηN*. Οι δομητες εκτελουνται απο αριστερα προς τα δεξια δηλ. δομητης-*βαση1*, δομητης-*βαση2* κλπ.

-- Παραδειγμα, Κληρονομικοτητα και εκτελεση Δομητων και Αποδομητων.

```
#include <iostream.h>
```

```
class Basi {
    public:
    Basi()  { cout << "Δομητης Βασης\n"; };
    ~Basi() { cout << "Καταστροφεας Βασης\n"; };
};

class Parag : public Basi {
    public:
    Parag()  { cout << "Δομητης παραγομενης\n"; };
    ~Parag() { cout << "Καταστροφεας παραγομενης\n"; };
};


int main()
{
    Parag ob;

    return(0);
}
// g43a.cpp //
```

Αποτελεσμα

```
Δομητης Βασης
Δομητης παραγομενης
Καταστροφεας παραγομενης
Καταστροφεας Βασης
```

Περασμα παραμετρων στον δομητη βασης

Ο δομητης βασης μπορει να κλειθει στην γραμμη αρχικοποιησεων (initialization line) του δομητη της παραγομενης ταξης. Οταν ο δομητης βασης απαιτει παραμετρους τοτε μπορουμε να περασουμε τις παραμετρους μεσω του δομητη παραγομενης. Χρησιμοποιουμε την συνθετη μορφη του δομητη παραγομενης

```
    δομητης_παραγομενης(παραμετροι) : βαση1(παραμετροι),
                                       βαση2(παραμετροι),
                                       ...
                                       βασηN(παραμετροι)
    {
        // σωμα δομητη παραγομενης
    }
```

Εδω η παραγομενη κληρονομει χαρακτηριστικα απο την *βαση1* εως και την *βασηN*.
Οι *παραμετροι* μπορει να ειναι σταθερες, μεταβλητες, παραστασεις, κλησεις σε συναρτησεις.

-- Παραδειγμα, Εκτελεσης δομητη βασης απο την γραμμη αρχικοποιησεων της παραγομενης.

```
#include <iostream.h>

class Basi {
    public:
    Basi() { cout << "Δομητης Βασης\n"; };
};
```

```
class Par : public Basi {
    public:
    Par() : Basi() { };  // Εκτελεση του δομητη βασης,
                         // και δομητη παραγομενης.
};

int main()
{
    Par A;     // Δηλωση αντικειμενου παραγομενης

    return(0);
}
// g1d.cpp //
```

-- Παραδειγμα, Περασματος παραμετρων στον δομητη βασης απο την δομητη παραγομενης.

```
#include <iostream.h>

class Basi {
    int x;
    public:
    Basi(int a) { x = a; };
    int getx() { return(x); };
};

class Parag : public Basi {
    int y;
    public:
    Parag(int b, int a) : Basi(a) { y = b; };
                        // στην βαση περνά η παραμετρος a
    int gety() { return(y); };
};

int main()
{
    Parag ob(4,15); // Το δευτερη ορισμα ειναι για την βαση

    int a,b;

    a = ob.getx();
    b = ob.gety();

    cout << a << "  " << b << endl;

    return(0);
}
// g44.cpp //
Αποτελεσμα
15  4
```

-- Περασμα παραμετρων σε δομητες βασεων οταν εχουμε κληρονομικοτητα πολλαπλων βασεων. Οι δομητες των δυο βασεων απαιτουν απο μια παραμετρο. Ο δομητης της παραγομενης δεν απαιτει παραμετρο.

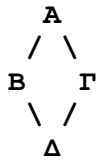
```
#include <iostream.h>

class Basi_A {
    int x;
    public:
    Basi_A(int i) { x = i; };
    int getx() { return(x); };
};

class Basi_B {
    int y;
    public:
    Basi_B(int k) { y = k; };
    int gety() { return(y); };
};

class Parag : public Basi_A, public Basi_B {
    public:
    Parag(int i, int k) : Basi_A(i), Basi_B(k) { };
};

int main()
{
    Parag ob(3,17);  // Η πρωτη παραμετρος προοριζεται για
                     // τον δομητη της
                     // Basi_A και η δευτερη για την Basi_B.
    int i,k;

    i = ob.getx();
    k = ob.gety();

    cout << i << "  " << k << endl;

    return(0);
}
// g45.cpp //
```

Αποτελεσμα

```
3  17
```

Εικονικη Βαση-Ταξη (`virtual` base class)

Σε μια κληρονομικοτητα οπως



η παραγομενη ταξη Δ κληρονομει δυο αντιγραφα της βασης Α, ενα απο την παραγομενη ταξη Β και ενα απο την παραγομενη ταξη Γ. Το επομενο προγραμμα λυνει το προβλημα με την χρηση του τελεστη `::` εμβελειας.

```
#include <iostream.h>

class Basi {
    public:
    int x;
};

class Par1 : public Basi{
    public:
    int y;
};

class Par2 : public Basi{
    public:
    int z;
};



class Parag : public Par1, public Par2 {
    public:
    int suma;
};

int main()
{
    Parag ob;

    // ob.x = 45;  ΛΑΘΟΣ. Το x ειναι ασαφες διοτι υπαρχουν
                //        δυο x, το ενα προερχεται απο το
                //        Par1 και το αλλο απο Par2

    ob.Par1::x = 5;  // ΟΚ με τον τελεστη εμβελειας ::
    ob.Par2::x = 7;

    ob.y = 30;
    ob.z = 40;

    ob.suma = ob.Par1::x + ob.y;
    cout << ob.suma << endl;

    ob.suma = ob.Par2::x + ob.z;
    cout << ob.suma << endl;

    return(0);
}
// g46.cpp //
```

Αποτελεσμα

```
35
47
```

Για να αποφυγουμε την υπαρξη πολλαπλων αντιγραφων της βασης σε ενα παραγομενο αντικειμενο δηλωνουμε την βαση-ταξη ως εικονικη (**virtual** base

class) οταν κληρονομειται. Με αυτον τον τροπο μονο ενα αντιγραφο της βασης υπαρχει στο παραγομενο αντικειμενο.

```
#include <iostream.h>

class Basi {
    public:
    int x;
};

class Par1 : virtual public Basi {  // Η βαση κληρονομειται
    public:                         // ως virtual
    int y;
};

class Par2 : virtual public Basi{  // Η βαση κληρονομειται
    public:                        // ως virtual
    int z;
};

class Parag : public Par1, public Par2 {
    public:
    int suma;
};

int main()
{
    Parag ob;

    ob.x = 10;  // Υπαρχει μονο ενα αντιγραφο της Βασης
                // (επειδη κληρονομηθηκε ως virtual)
                // Ισο με ob.Par1::x και ob.Par2::x
    ob.y = 3;
    ob.z = 4;

    ob.suma = ob.x + ob.y + ob.z;

    cout << ob.suma << endl;

    return(0);
}
// g47.cpp //
```

Αποτελεσμα
```
17
```

Δεικτες σε παραγομενους τυπους

Γενικα, ενας δεικτης ενος τυπου αντικειμενου δεν μπορει να δειξει σε αντικειμενο άλλου τυπου. Ο κανονας αυτος δεν εφαρμοζεται στις παραγομενες ταξεις. Ενας δεικτης βασης (base class pointer) μπορει να χρησιμοποιειθει να δειξει σε ενα αντικειμενο ταξης που παραγεται απο την βαση-ταξη, το αντιθετο δεν μπορει να γινει. Ο δεικτης βασης εχει προσβαση μονο σε μελη της παραγομενης ταξης που

προηλθαν (κληρονομιθηκαν) απο την βαση (εκτος και αν ο δεικτης βασης μετατραπει (cast) ωστε να εχει δικαιωματα σε ολη την παραγομενη ταξη).

-- Παραδειγμα. Δεικτες σε παραγομενους τυπους

```
#include <iostream.h>

class Basi {
    int x;
    public:
    void setx(int a) { x = a; };
    int getx() { return x; };
};

class Paragomeni : public Basi {
    int y;
    public:
    void sety(int a) { y = a; };
    int gety() { return y; };
};

int main()
{
    Basi *pB;
    Paragomeni P;

    pB = &P;     // Ο δεικτης βασης δειχνει σε παραγομενο
                 // αντικειμενο

    pB->setx(9); // Ο δεικτης βασης που δειχνει σε παραγομενο
                 // αντικειμενο εχει προσβαση σε μελη της
                 // βασης που κληρονομουνται.

    cout << pB->getx() << endl;

    // pB->sety(3); // ΛΑΘΟΣ. Ο δεικτης βασης δεν εχει
                    // προσβαση σε μελη της παραγομενης.

    Basi B;
    Paragomeni *pP;

    B.setx(12);

    // pP = &B; // ΛΑΘΟΣ. Ο δεικτης παραγομενης δεν μπορει να
                //        δειξει στην βαση (χωρις cast).

    pP = (Paragomeni *) &B; // Ο δεικτης παραγομενης μπορει
                // να δειξει στην βαση αφου τον κανουμε cast.

    cout << pP->getx() << endl;

    return(0);
}
// g17.cpp //
```

Εικονικες Συναρτησεις (`virtual` functions)

Μια συναρτηση δηλωνεται εικονικη οταν προηγειται η λεξη virtual. Εικονικη συναρτηση ειναι η συναρτηση που εχει δηλωθει ως εικονικη σε μια βαση-ταξη και ξαναοριζεται σε μια παραγομενη-ταξη. Ο ορισμος εκ νεου της συναρτησης στην παραγομενη-ταξη "υπερκαλυπτει" (overrides) την δηλωση της συναρτησης στην βαση-ταξη. Μια ταξη που περιλαμβανει μια εικονικη συναρτηση ονομαζεται πολυμορφηκη ταξη.

Στην πραγματικοτητα μια εικονικη συναρτηση οριζει γενικους ορους δρασης. Ενω η επαν-οροσμενη εικονικη συναρτηση υλοποιει (πραγματοποιει) μια ειδικη μεθοδο.

-- Στο επομενο παραδειγμα η συναρτηση vfn οριζεται ως εικονικη στην βαση. Η vfn ξαναοριζεται στις δυο παραγομενες ταξεις. Στο **`main`** ενας δεικτης βασης εκτελει την καταλληλη vfn αναλογα με το τυπο του αντικειμενου που δειχνει ο δεικτης βασης. Ανεξαρτητα απο το ποσες φορες η εικονικη συναρτηση κληρονομειται παραμενει εικονικη.

```
#include <iostream.h>

class Basi {
    public:
    virtual void vfn() { cout << "Η vfn της Basi\n"; };
             // η συναρτηση vfn οριζεται ως εικονικη
};

class Par1 : public Basi{
    public:
    void vfn() { cout << "Η vfn της Par1\n"; };
               // η vfn ξαναοριζεται
};

class Par2 : public Par1 {
    public:
    void vfn() { cout << "Η vfn της Par2\n"; };
               // η vfn ξαναοριζεται
};

int main()
{
    Basi   B,   // Αντικειμενο ταξης Basi
         *pB;   // Δεικτης σε ταξη Basi

    Par1 o1;    // Αντικειμενο o1 απο παραγομενη ταξη Par1
    Par2 o2;    // Αντικειμενο o2 απο παραγομενη ταξη Par2

    // Ο δεικτης βασης δειχνει στο B
    pB = &B;
    pB->vfn();  // εκτελεση της εικονικης της βασης

    // Ο δεικτης βασης δειχνει στο o1
    pB = &o1;
    pB->vfn();  // εκτελεση της εικονικης της παραγομενης
                // Par1
```

```
    // Ο δεικτης βασης δειχνει στο o2
    pB = &o2;
    pB->vfn();  // εκτελεση της εικονικης της παραγομενης
                // Par2

    return(0);
}
// g48.cpp //
```

Αποτελεσμα

```
H vfn της Basi
H vfn της Par1
H vfn της Par2
```

Οι συναρτησεις vfn στις παραγομενες μπορουν να κληθουν χρησιμοποιοντας το αντικειμενο πχ. o2.vfn(); με αυτον τον τροπο ομως δεν εχουμε πολυμορφισμο. Ο πολυμορφισμος κατα την εκτελεση (run time) επιτυγχανεται μονο μεσω ενος δεικτη βασης.

Αν και μοιαζουν διαφερουν μεταξυ των η υπερφορτωση (overloading) και η υπερκαλυψη (overriding) συναρτησεων. Το πρωτοτυπο μιας ξαναορισμενης εικονικης συναρτησης πρεπει να ειναι ακριβως το ιδιο οπως το πρωτοτυπο που οριζεται στην βαση.

Στατικες (**static**) και φιλες (**friend**) συναρτησεις μελη ταξης δεν μπορουν να ορισθουν εικονικες (**virtual**). Συναρτησεις δομητων δεν μπορουν να ειναι εικονικες, ενω οι συναρτησεις αποδομητων μπορουν.

Οι εικονικες συναρτησεις ειναι ιεραρχικες, οταν σε μια παραγομενη ταξη δεν "υπερκαλυπτεται" μια εικονικη συναρτηση τοτε χρησιμοποιειται η πρωτη επαν-ορισμενη συναρτηση που βρισκεται με αναποδη σειρα της ιεραρχειας.

-- Στο επομενο παραδειγμα η συναρτηση vfn οριζεται ως εικονικη στην βαση. Η vfn ξαναοριζεται μονο στην παραγομενη Par1. Η Par2 χρησιμοποιει την vfn της βασης επειδη παραγεται απ' αυτην.

```
#include <iostream.h>

class Basi {
    public:
    virtual void vfn() { cout << "H vfn της Basi\n"; };
             // η συναρτηση vfn οριζεται ως εικονικη
};

class Par1 : public Basi{
    public:
    void vfn() { cout << "H vfn της Par1\n"; };
            // η vfn ξαναοριζεται
};

class Par2 : public Basi {
    public:
    // Η vfn δεν ξαναοριζεται. Η vfn της Basi χρησιμοποιειται
    // επειδη η Par2 παραγεται απο την Basi
```

```
};

int main()
{
    Basi   B,   // Αντικειμενο ταξης Basi
         *pB;   // Δεικτης σε ταξη Basi

    Par1 o1;    // Αντικειμενο o1 απο παραγομενη ταξη Par1
    Par2 o2;    // Αντικειμενο o2 απο παραγομενη ταξη Par2

    // Ο δεικτης βασης δειχνει στο B
    pB = &B;
    pB->vfn();  // εκτελεση της εικονικης της βασης

    // Ο δεικτης βασης δειχνει στο o1
    pB = &o1;
    pB->vfn();  // εκτελεση της εικονικης της παραγομενης
                // Par1

    // Ο δεικτης βασης δειχνει στο o2
    pB = &o2;
    pB->vfn();  // εκτελεση της εικονικης της βασης

    return(0);
}
// g50.cpp //
```

Αποτελεσμα

```
Η vfn της Basi
Η vfn της Par1
Η vfn της Basi
```

-- Στο επομενο παραδειγμα η συναρτηση vfn οριζεται ως εικονικη στην βαση. Η vfn ξαναοριζεται στην παραγομενη Par1. Η Par2 χρησιμοποιει την vfn της Par1 επειδη παραγεται απ' αυτην.

```
#include <iostream.h>

class Basi {
    public:
    virtual vfn() { cout << "Η vfn της Basi\n"; };
              // η συναρτηση vfn οριζεται ως εικονικη
};

class Par1 : public Basi{
    public:
    void vfn() { cout << "Η vfn της Par1\n"; };
              // η vfn ξαναοριζεται
};

class Par2 : public Par1 {
    public:
    // Η vfn δεν ξαναοριζεται. Η vfn της Par1 χρησιμοποιειται
    // επειδη η Par2 παραγεται απο την Par1
};
```

```
int main()
{
    Basi   B,   // Αντικειμενο ταξης Basi
         *pB;   // Δεικτης σε ταξη Basi

    Par1 o1;    // Αντικειμενο o1 απο παραγομενη ταξη Par1
    Par2 o2;    // Αντικειμενο o2 απο παραγομενη ταξη Par2

    // Ο δεικτης βασης δειχνει στο B
    pB = &B;
    pB->vfn();  // εκτελεση της εικονικης της βασης

    // Ο δεικτης βασης δειχνει στο o1
    pB = &o1;
    pB->vfn();  // εκτελεση της εικονικης της παραγομενης
                // Par1

    // Ο δεικτης βασης δειχνει στο o2
    pB = &o2;
    pB->vfn();  // εκτελεση της εικονικης της Par1

    return(0);
}
// g49.cpp //
```

Αποτελεσμα

```
H vfn της Basi
H vfn της Par1
H vfn της Par1
```

Γνησιες Εικονικες (pure `virtual`) συναρτησεις

Μια "γνησια" (καθαρη - αγνη) εικονικη συναρτηση ειναι μια εικονικη συναρτηση που δεν εχει "σωμα" (δεν εχει εντολες) μεσα την βαση. Η γενικη μορφη δηλωσης γνησιας εικονικης συναρτησης ειναι:

```
virtual τυπος_συναρτησης ονομα_συναρτησης(παραμετροι) = 0;
```

Οταν μια εικονικη συναρτηση ειναι γνησια τοτε σε ολες τις παραγομενες ταξεις η εικονικη συναρτηση πρεπει να ξαναορισθει (σε αντιθετη περιπτωση εχουμε λαθος στην διαρκεια της μεταφρασης).

-- Παραδειγμα Γνησιας εικονικης συναρτησης.

```
#include <iostream.h>

class Num {
  protected:
    int x;
  public:
    setx(int a) { x = a; };

    virtual void disp() = 0;  // δηλωση γνησιας εικονικης
                              // συναρτησης
```

```
};

class Dec : public Num {
  public:
    void disp() { cout << x << endl; };  // δεκαδικο συστημα
};

class Oct : public Num {
  public:
    void disp() { cout << oct << x << endl; }; // οκταδικο
};

class Hex : public Num {
  public:
    void disp() { cout << hex << x << endl; }; // δεκαεξαδικο
};

int main()
{
    Num *pB;   // Δεικτης στην βαση ταξη Num

    Dec d;    // Αντικειμενο d απο παραγομενη ταξη Dec
    Oct o;    // Αντικειμενο o απο παραγομενη ταξη Oct
    Hex h;    // Αντικειμενο h απο παραγομενη ταξη Hex

    // Ο δεικτης βασης δειχνει στο d
    pB = &d;
    pB->setx(27);
    pB->disp();  // εκτελεση της εικονικης της παραγομενης
                 // Dec

    // Ο δεικτης βασης δειχνει στο o
    pB = &o;
    pB->setx(27);
    pB->disp();  // εκτελεση της εικονικης της παραγομενης
                 // Oct

    // Χρησιμοποιηση του αντικειμενου h
    h.setx(27);
    h.disp();  // εκτελεση της εικονικης της παραγομενης Hex

    return(0);
}
// g51.cpp //
```

Αποτελεσμα

```
27
33
1b
```

Αφηρημενες Ταξεις (abstract classes)

Οταν μια ταξη εχει εστω και μια γνησια εικονικη (pure virtual) συναρτηση τοτε λεγεται αφηρημενη ταξη. Μια αφηρημενη ταξη αποτελει εναν ατελη τυπο και χρησιμοποιειται ως θεμελιο για τις παραγομενες ταξεις. Απο μια αφηρημενη ταξη

δεν μπορουν να δημιουργηθουν αντικειμενα. Μπορουν να δημιουργηθουν δεικτες και αναφορες για μια αφηρημενη ταξη. Κατ' αυτον τον τροπο οι αφηρημενες ταξεις υποστηριζουν τον πολυμορφισμο κατα το run-time (ο δεικτης βασης διαλεγει την καταλληλη εικονικη συναρτηση)

Μια βασικη ιδεα στον αντικειμενοστρεφη προγραμματισμο ειναι η αρχη "one interface, multiple methods" Στην C++, μια βαση ταξη μπορει να χρησιμοποιειθει για να ορισθει η φυση του interface γενικα. Στην συνεχεια σε καθε παραγομενη ταξη πραγματοποιουνται οι "ειδικες" λειτουργιες. Ενας τροπος (δυνατος και ευελικτος) για να πραγματοποιειθει το "one interface, multiple methods" ειναι η χρηση εικονικων συναρτησεων,αφηρημενων τυπων, και ο run-time πολυμορφισμος.

-- Στο παρακατω προγραμμα δημιουργειται μια ιεραρχεια απο ταξεις που κανει μετατροπες μεταξυ διαφορετικων συστηματων μετρισης (πχ. απο μιλια σε χιλιομετρα) Δυο μεταβλητες στην βαση Conv αποθηκευουν την αρχικη τιμη (arx) την τιμη (met) που προερχεται απο την μετατροπη. Η μεταπροπη γινεται απο μια γνησια εικονικη συναρτηση ypol η οποια ξαναοριζεται στις δυο παραγομενες MtoK και FtoC. Αν και η ypol (μεθοδος) διαφερει στις δυο παραγομενες το interface παραμενει το ιδιο.

```
#include <iostream.h>

class Conv {
    public:
    Conv(float a) { arx = a; };

    float geta() { return(arx); };
    float getm() { return(met); };

    virtual void ypol() = 0;  // pure virtual συναρτηση

    protected:
    float arx;   // αρχικη τιμη
    float met;   // μετατροποποιημενη τιμη

};

class MtoK : public Conv {   // μιλια σε χιλιομετρα
    public:
    MtoK(float a) : Conv(a) {};
    void ypol() { met = arx*1.653; };
};

class FtoC : public Conv {   // φαρενάιτ σε κελσιου
    public:
    FtoC(float a) : Conv(a) {};
    void ypol() { met =  (arx - 32)/1.8; };
};

int main()
{
    Conv *pB;     // Δεικτης στην βαση ταξη conv
```

```
    MtoK mk(10);  // Αντικειμενο mk παραγόμενης ταξης MtoK
    FtoC fc(69);  // Αντικειμενο fc παραγόμενης ταξης FtoC
    //
    pB = &mk;     // Ο δεικτης βασης δειχνει στο mk
    pB->ypol();   // Εκτελεση της εικονικης ypol της MtoK

    cout << mk.geta() << " μιλια ειναι " << mk.getm()
         << " Km\n";
    //
    pB = &fc;     // Ο δεικτης βασης δειχνει στο fc
    pB->ypol();  // Εκτελεση της εικονικης ypol της FtoC

    cout << fc.geta() << " βαθμοι Φαρενάιτ ειναι "
         << fc.getm() << " βαθμοι Κελσιου" << endl;

    return(0);
}
// g52.cpp //
```

Αποτελεσμα

```
10 μιλια ειναι 16.53 Km
69 βαθμοι Φαρενάιτ ειναι 20.5556 βαθμοι Κελσιου
```

-- Στο παραπανω προγραμμα να προσθεθει μια παραγομενη ταξη που μετατρεπει ιντσες σε εκατοστομετρα.

Λυση

```
class ItoE : public Conv {   // Ιντσες σε εκατοστομετρα cm
    public:
    ItoE(float a) : Conv(a) {};
    void ypol() { met = 2.7*arx; };
};
```

Πολυμορφισμος (Polymorfism)

Ο πολυμορφισμος υποστηριζεται απο την C++ και στην διαρκεια της μεταφρασης (compile time ή early binding) και στην διαρκεια εκτελεσης (run time ή late binding). Ο πολυμορφισμος κατα την διαρκεια της μεταφρασης υποστηριζεται με υπερφοτωση συναρτησεων και τελεστων. Ο πολυμορφισμος κατα την εκτελεση επιτυγχανεται με την κληρονομικοτητα (inheretance) και της εικονικες συναρτησεις (virtual functions)

Συνδεση συναρτησεων (binding)

Κατα την διαρκεια της μεταφρασης ολες οι πληροφοριες για τις κλησεις των συναρτησεων ειναι γνωστες και τα αντικειμενα "δενονται" με τις συναρτησεις (**early binding**). Οι κλησεις "κοινων" συναρτησεων οπως της standard βιβλιοθηκης, οι κλησεις συναρτησεων με υπερφοτωση , οι κλησεις σε συναρτησεις υπερφοτωσης τελεστων δενονται με τα αντικειμενα κατα την μεταφραση. Πλεονεκτημα, η κληση των συναρτησεων γινεται με ταχυτητα.

Οι εικονικες συναρτησεις (οταν χρησιμοποιειται δεικτης βασης) "δενονται" με τα αντικειμενα κατα την διαρκεια εκτελεσης (**late binding**) του προγραμματος.

Πλεονεκτημα, η δημιουργια ευεληκτων προγραμματων που μπορουν πχ. να ανταποκριθουν σε γεγονοντα (events) που συμβαινουν κατα την διαρκεια εκτελεσης του προγραμματος. Επειδη ολες οι πληροφοριες ειναι αγνωστες και πρεπει να βρεθουν (υπολογισθουν) για γινει μια κληση σε συναρτηση μπορει η εκτελεση του προγραμματος να ειναι κατά τι πιο αργη.

Προτυπα (Templates)

Τα προτυπα επιτρεπουν τον ορισμο **γενικων συναρτησεων και ταξεων**, οπου ο τυπος των δεδομενων πανω στα οποια η συναρτηση ή η ταξη λειτουργει οριζεται ως μια παραμετρο.

Γενικες συναρτησεις (generic functions)

Σε μια γενικη συναρτηση ο τυπος των δεδομενων (τα οποια η συναρτηση επεξεργαζεται) περνά ως παραμετρο.

Πρωτοτυπο:

```
template <class Τυπος> τυπος ονομα_συναρτησης(παραμετροι);
```

Υλοποιηση:

```
template <class Τυπος> τυπος ονομα_συναρτησης(παραμετροι)
{
    // σωμα
}
```

(η παραμετρος **Τυπος** ειναι ο τυπος των δεδομενων)

Για να ορισθει ενα προτυπο (**template**) συναρτησης πρεπει μια τουλαχιστον απο τις παραμετρους να ειναι **Τυπος**.

```
α) template <class T> int  fn(T &a);
β) template <class X> void f(int b, X &g);
γ) template <class T> T hoo(T g, int a, char b);
δ) // template <class Y> Y foo(float a);
```

ΛΑΘΟΣ. Δεν εχει κανενα ορισμα τυπου **Y**.

Οι γενικες συναρτησεις ειναι οπως οι υπερφορτωμενες συναρτησεις αλλα εχουν περισσοτερους περιορισμους. Σε μια γενικη συναρτηση μονο ο τυπος των δεδομενων επιτρεπεται να διαφερει, ο κωδικας ειναι ο ιδιος για ολες τις εκδοχες (version) της συναρτησης.

-- Στο παρακατω προγραμμα οριζεται το προτυπο συναρτησης swap. Η συναρτηση αλλαζει αμοιβαια τις τιμες σε δυο μεταβλητες. Ο τυπος δεδομενων (μεταβλητων) που η συναρτηση θα λειτουργει περνα ως παραμετρο T και μπορει να ειναι **int**, **char** κλπ.

```
#include <iostream.h>
template <class T> void swap(T &a, T &b)
{
    T temp;

    temp = a;
    a = b;
    b = temp;
}

int main()
{
    int x = 2, y = 4;
    swap(x,y);        // κληση της swap με τυπο δεδομενων int
    cout << x << "  " << y << endl;

    double d = 5.3, f = 6.8;
    swap(d,f);     // κληση της swap με τυπο δεδομενων double
    cout << d << "  " << f << endl;

    char c = 'R', v = 'w';
    swap(c,v);       // κληση της swap με τυπο δεδομενων char
    cout << c << "  " << v << endl;

    // swap(x,d); ΛΑΘΟΣ. Συνδιασμος int και double

    return(0);
}
// g53.cpp //
```

Αποτελεσμα

```
4    2
6.8  5.3
w    R
```

Ο μεταφραστης αυτοματα αντικαθιστα τον γενικο τυπο T με τους πραγματικους τυπους των παραμετρων και δημιουργει τρια αντιγραφα της swap (ενα για να αλλαζει ακεραιες τιμες, ενα για αλλαζει πραγματικες τιμες, και ενα για να αλλαζει χαρακτηρες.

-- Στο παρακατω προγραμμα οριζεται μια προτυπη συναρτηση disp με δυο γενικους τυπους X και Y. Η συναρτηση disp εμφανιζει το περιεχομενο δυο μεταβλητων.

```
#include <iostream.h>

template <class X, class Y>
void disp(X a, Y b)
{
    cout << a << "   " << b << endl;
}
```

```
int main()
{
    int x = 2;  char *s = "Dream";

    disp(x,s);       // Εμφανιση int και string

    disp(5.9,'U');   // Εμφανιση double και char

    return(0);
}
// g54.cpp //
```

Αποτελεσμα

```
2   Dream
5.9   U
```

Υπερφορτωση γενικης συναρτησης

Σε ενα προγραμμα μπορει μια γενικη (προτυπη) συναρτηση να υπερφορτωθει με μια "ειδικη" συναρτηση που υπερκαλυπτει την γενικη συναρτηση.

```
#include <iostream.h>

template <class T>    // γενικη για ολους τους τυπους
void swap(T &a, T &b)
{
    T temp;

    temp = a;
    a = b;
    b = temp;
}

void swap(int &a, int &b)  // ειδικη μονο για int
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}

int main()
{
    int x = 2, y = 4;
    swap(x,y);     // κληση της ειδικης swap με τυπους int
    cout << x << "  " << y << endl;

    double d = 5.3, f = 6.8;
    swap(d,f);     // κληση της γενικης swap με τυπους double
    cout << d << "  " << f << endl;

    char c = 'R', v = 'w';
```

```
    swap(c,v);      // κληση της γενικης swap με τυπους char
    cout << c << "  " << v << endl;

    return(0);
}
// g55.cpp //
```

Αποτελεσμα

```
4    2
6.8  5.3
w    R
```

Ο μεταφραστης δημιουργει δυο αντιγραφα της γενικης swap (ενα για να αλλαζει πραγματικες τιμες, και ενα για να αλλαζει χαρακτηρες). Για τις ακεραιες τιμες καλει την υπερφορτωμενη ειδικη swap που οριζεται στο προγραμμα.

Γενικες Ταξεις (generic classes)

Στις γενικες ταξεις οριζονται ολοι οι αλγοριθμοι που χρησιμοποιουνται απο αυτην την ταξη αλλα ο τυπος των δεδομενων που επεξεργαζονται οριζεται ως παραμετρο οταν αντικειμενα αυτης της ταξης δημιουργουνται. Μια γενικη ταξη δηλωνεται ως εξης

```
template <class Ττυπος> class ονομα_ταξης {
    ...
};
```

(η παραμετρος ***Ττυπος*** ειναι ο τυπος των δεδομενων)

Ενα αντικειμενο μιας γενικης ταξης δηλωνεται ως εξης

```
ονομα_ταξης<τυπος> ονομα_αντικειμενου;
```

(Ο ***τυπος*** μπορει να ειναι απλος (**int**, **char** κλπ) ή αντικειμενο μιας ταξης)

-- Παραδειγμα δηλωσης μια γενικης ταξης με δυο συναρτησεις μελη. Στο κυριως προγραμμα οταν δημιουργουνται τα αντικειμενα (a, d και c) της γενικης ταξης περνα ως παραμετρο ο τυπος των δεδομενων που η γενικη ταξη θα επεξεργασθει.

```
#include <iostream.h>

template <class T> class Obj {   // γενικη ταξη
    T x;
    public:
    void setx(T a);
    T getx();
};

template <class T>       // συναρτηση μελος γενικης ταξης
void Obj<T>::setx(T a)
{
    x = a;
}

template <class T>       // συναρτηση μελος γενικης ταξης
```

```
T Obj<T>::getx()
{
    return(x);
}

int main()
{
    Obj<int> a;// Αντικειμενο a τυπου Obj. Ο τυπος δεδομενων
               // που η γενικη ταξη θα λειτουργησει ειναι int
    a.setx(15);
    cout << a.getx() << endl;

    Obj<double> d; // Αντικειμενο d τυπου Obj.  Ο τυπος
                   // δεδομενων γενικης ταξης ειναι double
    d.setx(6.8);
    cout << d.getx() << endl;

    Obj<char> c;   // Αντικειμενο c τυπου Obj.  Ο τυπος
                   // δεδομενων γενικης ταξης ειναι char
    c.setx('Q');
    cout << c.getx() << endl;

    return(0);
}
// g56.cpp //
```

-- Παραδειγμα δηλωσης μια γενικης ταξης με δυο παραμετρους. Στο κυριως προγραμμα οταν δημιουργουνται τα αντικειμενα (A και B) της γενικης ταξης περνανε ως παραμετροι οι δυο τυποι των δεδομενων που η γενικη ταξη θα επεξεργασθει.

```
#include <iostream.h>

template <class X, class Y>  // γενικη ταξη με δυο παραμετρους
class Obj {
    X i;
    Y k;
    public:
    Obj(X a, Y b);
    void getik(X &a, Y &b);
    X geti() { return(i); };
    Y getk();
};

template <class X, class Y> Obj<X,Y>::Obj(X a, Y b)// Δομητης
{
    i = a;
    k = b;
}

template <class X, class Y>        // συναρτηση μελος
void Obj<X,Y>::getik(X &a, Y &b)
{
    a = i;
    b = k;
}
```

```
// Συναρτηση μελος
template <class X, class Y> Y Obj<X,Y>::getk()
{
    return(k);
}

int main()
{
    Obj<int,char> A(12,'Z'); // Αντικειμενο Α τυπου Obj.
                  // Οι τυποι δεδομενων που η γενικη ταξη θα
                  // λειτουργησει ειναι int και char
    cout << A.geti() << "  " << A.getk() << endl;

    Obj<long,double> B(5L,3.1); // Αντικειμενο Β τυπου Obj.
                  // Οι τυποι δεδομενων που η γενικη ταξη θα
                  // λειτουργησει ειναι long και double
    cout << B.geti() << "  " << B.getk() << endl;

    long q; double w;
    B.getik(q,w);
    cout << q << "  " << w << endl;

    return(0);
}
// g57.cpp //
```

Αποτελεσμα

```
12  Z
 5  3.1
 5  3.1
```

Χειρισμος Εξαιρεσεων (exception handling)

Ο χειρισμος εξαιρεσεων μας επιτρεπει να χειριζομαστε τα λαθη που προκυπτουν κατα την εκτελεση (run time errors) ενος προγραμματος. Ο χειρισμος εξαιρεσεων στην C++ χρησιμοποιει τρεις λεξεις κλειδια:

try, **catch**, και **throw**

Τις εντολες που θελουμε να παρακολουθησουμε (για τυχον λαθη στην διαρκεια της εκτελεσης του προγραμματος) τις τοποθετουμε σε ενα **try** μπλοκ. Εαν παρουσιασθει μια εξαιρεση μεσα στο **try** μπλοκ (δηλ. ενα λαθος) τοτε το "πεταμε" χρησιμοποιοντας την **throw**. Η εξαιρεση "πιανεται" με την **catch** και επεξεργαζεται καταλληλως.
Ο κωδικας μεσα στο **catch** εκτελειται μονο οταν εκτελειται ενα **throw**.

-- Ενα συνηθες λαθος (run-time error) κατα την εκτελεση των προγραμματων ειναι η διαιρεση δια του μηδενος. Για να το αποφυγουμε κανουμε χρηση της εξαιρεσης.

```
#include <iostream.h>

int main()
{
    double x,y;

    cout << "Δωσε τον διαιρεταιο ";
    cin >> x ;

    cout << "Δωσε τον διαιρετη ";
    cin >> y ;

    try {
        if ( y == 0 )  // Εαν ο διαιρετης = 0 τοτε
            throw 100; // "πέτα" το λαθος με ακεραια τιμη 100
                       // (το catch δεχεται ακεραιες τιμες)
        cout << "x/y = " << x/y << endl;
    }

    catch(int a) {
        if (a == 100)
            cout << "Αδυνατη διαιρεση οταν ο διαιρετης=0\n";
    }

    return(0);
}
// g58.cpp //
```

Παρατηρηση Στο **throw 100** αντι του 100 θα μπορουσαμε να βαλουμε εναν οποιοδηποτε ακεραιο.

-- Το επομενο προγραμμα διαβαζει δυο ακεραιους αριθμους (αντιπροσωπευουν τις ηλικιες πατερα και υιου), υπολογιζει και εμφανιζει τον λογο ηληκιας του πατερα προς την ηλικια του υιου. Η ηλικια του υιου δεν μπορει να ειναι 0 (μηδεν) ως επισης και οι ηλικιες του πατερα και του υιου δεν επιτρεπεται να ειναι αρνητικες. Δια να

προσταθευθει ο χρηστης απο τυχον λαθη κατα την εισαγωγη των ηλικιων κανουμε χρηση των εξαιρεσεων.

```
#include <iostream.h>

int main()
{
   int father,son;

   cout << "Ηλικια πατερα: ";
   cin >> father;
   cout << "Ηλικια υιου: ";
   cin >> son;

   try {
      if ( son == 0 ) {
         throw 100;
      }

      if ( son > father) {
         throw 101;
      }

      if ( father <= 0 || son < 0) {
         throw 198;
      }

      cout << "Ο πατερας ειναι μεγαλυτερος απο τον υιο "
           << father/float(son) << " φορες " << endl;
   }

   catch(int a) {
      if (a == 100)
         cout << "Η ηλικια του υιου δεν μπορει να ειναι 0."
              << endl;
      if (a == 101)
         cout << "Η ηλικια του υιου δεν μπορει να ειναι "
              << "μεγαλυτερη απο την ηλικια του πατερα."
              << endl;
      if(a == 198)
         cout << "Οι ηλικιες πρεπει να ειναι θετικοι "
              << "αριθμοι" << endl;
   }
   return(0);
}
// g95.cpp //
```

-- Για να αποφυγουμε την διαιρεση δια του μηδενος κανουμε χρηση της εξαιρεσης. Στο επομενο προγραμμα η διαιρεση και ολος ο μηχανισμος της εξαιρεσης τοποθετειται σε συναρτηση.

```
#include <iostream.h>

void diairesi(double a, double b);
```

```
int main()
{
    double x,y;

    cout << "Δωσε τον διαιρεταιο ";
    cin >> x ;

    cout << "Δωσε τον διαιρετη ";
    cin >> y ;

    diairesi(x,y);

    return(0);
}

void diairesi(double a, double b)
{
   try {
      if ( b == 0 )  // εαν ο διαιρετης = 0 τοτε "πέτα" το
                     // λαθος
         throw 100;  // με τιμη 100

      cout << "a/b = " << a/b << endl;
   }
   catch(int a) {
      if (a == 100)  // το λαθος εχει τιμη 100
         cout << "Αδυνατη διαιρεση οταν ο διαιρετης = 0\n";
   }
}
// g59.cpp //
```

Μια εξαιρεση μπορει να "πεταχθει" απο μια συναρτηση αρκει η συναρτηση να εχει κλειθει μεσα απο ενα **try** μπλοκ.

```
#include <iostream.h>

void exeresi(double a);

int main()
{

   try {
      exeresi(0);
      exeresi(1);
      exeresi(2);
   }

   catch(double d) {
      cout << "Ανινχνευθηκε εξαιρεση με τιμη " << d << endl;
   }

   return(0);
}
```

```
void exeresi(double a)
{
   cout << "a = " << a << endl;

   if ( a == 1 )
      throw a;

}
// g61.cpp //
```

Αποτελεσμα

```
a = 0
a = 1
Ανιχνευθηκε εξαιρεση με τιμη 1
```

Ο κωδικας στο **catch** δεν εκτελειται οταν το a ειναι μηδεν.

```
#include <iostream.h>

void exeresi(int a);

int main()
{
   exeresi(-1);
   exeresi(0);
   exeresi(1);
   exeresi(2);

   return(0);
}

void exeresi(int a)
{
   try {
      if (a) throw a;
   }
   catch(int x) {
      cout << "Εξαιρεση # " << a << endl;
   }
}
// g62.cpp //
```

Αποτελεσμα

```
Εξαιρεση # -1
Εξαιρεση # 1
Εξαιρεση # 2
```

Μπορουμε να εχουμε περισσορερα απο ενα **catch** σε καθε **try**. Καθε **catch** "πιανει" διαφορετικο τυπο εξαιρεσης. Το επομενο προγραμμα "πιανει" εξαιρεσεις με ακεραιους και στρινγ.

```
#include <iostream.h>

void exeresi(int a);

int main()
{
   exeresi(-1);
   exeresi(0);
   exeresi(1);
   exeresi(2);

   return(0);
}

void exeresi(int a)
{
   try {
      if (a)
          throw a;          // "πετά" ενα ακεραιο
      else
          throw "a = 0";    // "πετά" ενα στρινγ
   }

   catch(int x) {
      cout << "Εξαιρεση # " << a << endl;
   }

   catch(char *s) {
      cout << "Εξαιρεση string " << s << endl;
   }

}
// g63.cpp //
```

Ελλειπτικη συνταξη του `catch`

Εαν θελουμε η **`catch`** να "πιανει" ολες τις εξαιρεσεις τοτε χρησιμοποιουμε την ελλειπτικη συνταξη του **`catch`**

```
catch(...) {
    // εντολες
}
```

-- Παραδειγμα ελλειπτικου **`catch`**

```
#include <iostream.h>

void exeresi(int a);

int main()
{
   exeresi(-1);
   exeresi(0);
   exeresi(1);
   exeresi(2);
```

```
   return(0);
}

void exeresi(int a)
{
   try {
      if (a == -1) throw 6.7;
      if (a == 0)  throw a;
      if (a == 1)  throw 'z';
      if (a == 2)  throw 5L;
   }
                  // Ελλειπτικο catch(...)
   catch(...) {    // "πιανει" ολες τις εξαιρεσεις
      cout << "Εξαιρεση (a=" << a << ")" << endl;
   }
}
// g65.cpp //
```

Αποτελεσμα

```
Εξαιρεση (a=-1)
Εξαιρεση (a=0)
Εξαιρεση (a=1)
Εξαιρεση (a=2)
```

-- Στο παρακατω προγραμμα ενα **catch(int)** πιανει ολες τις **int** εξαιρεσεις και ενα ελλειπτικο **catch(...)** πιανει ολες τις υπολοιπες εξαιρεσεις.

```
#include <iostream.h>

void exeresi(int a);

int main()
{
   exeresi(-1);
   exeresi(0);
   exeresi(1);
   exeresi(2);

   return(0);
}

void exeresi(int a)
{
   try {
      if (a == -1) throw 6.7;
      if (a == 0)  throw a;
      if (a == 1)  throw 'z';
      if (a == 2)  throw 5L;
   }
   catch(int x) {
      cout << "Εξαιρεση int" << endl;
   }
   catch(...) {    // πιανει ολους του τυπους των εξαιρεσεων
      cout << "Εξαιρεση αλλων τυπων" << endl;
   }
```

```
}
// g64.cpp //
```

Αποτελεσμα

```
Εξαιρεση αλλων τυπων
Εξαιρεση int
Εξαιρεση αλλων τυπων
Εξαιρεση αλλων τυπων
```

Περιορισμος Εξαιρεσεων (restricting exceptions)

Οταν μια συναρτηση καλειται μεσα απο ενα **try** μπλοκ και η συναρτηση χειριζεται **throw** τοτε η συναρτηση μπορει να χειρισθει τους τυπους εξαιρεσεων που ο προγ/στης θελει.

```
τυπος ονομα_συναρτησης(ορισματα) throw(τυποι_εξαιρεσεων)
{
    // ...
}
```

-- Παραδειγμα

```
#include <iostream.h>

void exeresi(int a) throw(int,char); // πρωτοτυπο

int main()
{
    try {
        exeresi(0);   // exeresi(1);
    }
    catch(int i) {
        cout << "Εξαιρεση int" << endl;
    }
    catch(char ch) {
        cout << "Εξαιρεση char" << endl;
    }

    return(0);
}

void exeresi(int a) throw(int,char)
{
    if (a == 0)  throw a;     // int
    if (a == 1)  throw 'z';   // char
}
// g66.cpp //
```

Ειναι δυνατον να "ξαναπεταξουμε" μια εξαιρεση εκτελωντας ενα **throw**. Ετσι η εξαιρεση διαβιβαζεται σε μια πιο "κεντρικη" **try**/**catch** ακολουθια.

```
#include <iostream.h>
```

```
void exeresi(int a);



int main()
{
    try {
       exeresi(1);
    }
    catch(char *) {
       cout << "Εξαιρεση στρινγ" << endl;
    }

    return(0);
}

void exeresi(int a)
{
   try {
      if (a == 1)  throw "acid rain";
                // "πέτα" την εξαιρεση (στρινγ)
   }
   catch(char *s) {
      throw;    // "ξαναπέτα" την εξαιρεση
   }
}
// g67.cpp //
```

Τυπος bool

Οι μεταβλητες τυπου **bool** μπορουν να λαβουν μια Boolean τιμη **true** ή **false.**

```
#include <iostream.h>
int main()
{
    bool a = true;

    if(a == false)
        cout << "Τιμη = " << a << endl;
    else
        cout << "Τιμη = " << a << endl;

    return(0);
}
```

Αποτελεσμα

```
Τιμη = true
```

Συναρτησεις μετατροπων

Συναρτησεις μετατροπων (conversion functions) ειναι συναρτησεις που απλως μετατρεπουν ενα τυπο (αντικειμενο μιας ταξης) σε εναν αλλο τυπο. (Εαν ενα αντικειμενο μιας ταξης περιλαμβανεται σε μια παρασταση τοτε χρησιμοποιουμε υπερφορτωση τελεστων)
Η γενικη μορφη συναρτησεων μετατροπης ειναι

```
    operator τυπος() { return τιμη; };
```

-- Παραδειγμα μετατροπης αντικειμενων τυπου **Obj** σε τυπο **int**.

```
#include <iostream.h>
#include <string.h>

class Obj {
    int x;      // Μηκος του στρινγ
    char *str;  // Δεικτης στο στρινγ
    public:
    Obj(char *s) { str = s;  x = strlen(s); }; // Δομητης
    char *getstr() { return(str);};
    operator int() { return(x); }; // Συναρτηση μετατροπης
};                                 // τυπου Obj σε τυπο int

int main()
{
    Obj A("cage");
    int z;

    z = A;  // Μετατροπη Obj σε int
            // Ιδιο με: z = A.operator int();

    cout << "Το στρινγ " << A.getstr()
```

```
        << " εχει μηκος " <<  z << endl;

    return(0);
}
// g68.cpp //
```

Αποτελεσμα

```
Το στρινγ cage εχει μηκος 4
```

Παρατηρηση. Το `z = A;` μπορει να γραφει και ως

```
z = A.operator int();
```

Συνδεση C με C++

Μπορουμε να συνδεσουμε (link) στα C++ προγραμματα μας συναρτησεις που εχουμε γραψει σε C ή σε αλλη γλωσσα προγραμματισμου. Η γενικη μορφη συνδεσης ειναι

```
extern "γλωσσα" πρωτοτυπα_συναρτησεων
```

-- Παραδειγμα συνδεσης μιας C συναρτησης με ενα C++ προγραμμα. (Εχει χρησιμοποιηθει ο μεταφραστης της GNU **C++** για DOS)

```
/*  C συναρτηση.
    Ονομα πηγαιου προγραμματος g70.c  */

void c_synartisi()
{
    printf("C συναρτηση\n");
}
```

```
/*  Δημιουργια object προγραμματος g70.o με την εντολη
    gcc -c g70.c  */
```

```
//  C++ προγραμμα  g71.cpp

#include <iostream.h>

extern "C" {     // Χρηση συναρτησεων γραμμενες σε γλωσσα C
   void c_synartisi(); // Πρωτοτυπο συναρτησης
}

int main()
{
   c_synartisi();      // Κληση της C συναρτησης

   return(0);
}
```

Δημιουργια εκτελεσιμου προγραμματος **a.exe** με την εντολη

```
gxx -o a.exe g70.o g71.cpp
```

Τελεστες μετατροπης τυπων

Εκτος απο τον γνωστο τροπο μετατροπης (cast) τυπων υπαρχουν τεσσερις νεοι τελεστες στην ANSI C++

1. const_cast

```
    2. dynamic_cast
    3. reinterpret_cast
    4. static_cast
```

Η γενικη μορφη ειναι:

```
    τελεστης<τυπος> (αντικειμενο)
```

Ο τελεστης **const_cast** χρησιμοποιειται για να υπερκαλυψει το **const** και/ή το **volatile** σε μια μετατροπη (χρηση: αφαιρει το **const** απο ενα αντικειμενο).

Ο τελεστης **dynamic_cast** εκτελει και ελεγχει μια μετατροπη κατα την εκτελεση (run-time). Εαν η μετατροπη ειναι αδυνατη τοτε η παρασταση εκτιμαται σε null (χρηση: μετατροπες πολυμορφικων τυπων).

Ο τελεστης **reinterpret_cast** αλλαξει εναν τυπο σε εναν αλλο τυπο (χρηση: μετατροπη ασυμβατων τυπων).

Ο τελεστης **static_cast** εκτελει μη-πολυμορφικη μετατροπη (χρηση: α) μετατροπη ενος δεικτη βασης σε δεικτη παραγομενης ταξης, β) συνηθεις μετατροπες.

Μονο ο τελεστης **const_cast** αφαιρει το **const** απο ενα αντικειμενο.

-- Προγραμμα μετατροπης δεικτη σε χαρακτηρα (**char ***) σε ακεραιο (**short int**)

```
#include <iostream.h>

int main()
{
    char *p = "zoo";
    short int i;

    i = reinterpret_cast<int> (p);

    cout << i << endl;

    return(0);
}
// g72.cpp //
```

-- Παραδειγμα με τους τελεστες μετατροπης τυπων.

```
#include <iostream.h>

int main(void)
{
    // const_cast.  Προσωρινη "αφαιρεση" του const
    const long k = 34;
    // ΛΑΘΟΣ // long *p = &k;
    long *pi = const_cast<long *> (&k);
    cout << *pi << endl;

    // static_cast
    int j = 32;
    long l = static_cast<long>(j);
```

```
    cout << l << endl; //


    // reinterpret_cast.  Μετατροπη δεικτη χαρακτηρα (char *)
    // σε ακεραιο (short int)

    char *p = "zoo";
    short int i;
    i = reinterpret_cast<int> (p);
    cout << i << endl;

    return(0);
}
```

-- Παραδειγμα τελεστη **dynamic_cast**

```
// Τεστ με gcc: gxx -frtti g83.cpp
#include <typeinfo>
#include <iostream.h>

class B {            // Ταξη B με εικονικη συναρτηση
    public: virtual ~B(){};
};

class D : public B {};  // Παραγομενη D
class E : public B {};  // Παραγομενη E

void func(B* pB)
{
    D* pD = dynamic_cast<D*>(pB);
    if (pD)
        cout << "Ο pB δειχνει σε ενα αντικειμενο τυπου D\n";
    else
        cout << "pD = " << pD << endl;  // pD = 0

}

int main(void)
{
    B oB,*pB;
    pB = &oB;
    func(pB);      //  pD = 0x0 (hex)

    D oD;
    pB = &oD;
    func(pB);      // pD δειχνει σε ενα D

    return(0);
}
// g83.cpp //
```

Αποτελεσμα

```
pD = 0x0
Ο pB δειχνει σε ενα αντικειμενο τυπου D
```

Ορισμος χωρου με namespace

Με το **namespace** οριζουμε ενα block (χωρο) με αντικειμενα (μεταβλητες και συναρτησεις) κατω απο ενα ονομα.
Η γενικη μορφη ειναι

```
namespace ονομα_χωρου {
     // δηλωσεις_αντικειμενων
}
```

Παραδειγμα

```
namespace mysp {
    int x,y;
    void myfn(int a) { cout << a << endl; };
}

// Η προσβαση σε μεταβλητες του mysp γινεται με τον
// τελεστη εμβελειας ::

mysp::x = 12;
mysp::myfn(13);
```

Εντολη using

Οταν ενα μελος του χωρου χρησιμοποιειται συχνα τοτε μπορουμε να κανουμε χρηση της **using** για να απλοποιησουμε την προσβαση του.
Η εντολη **using** εχει δυο μορφες:

A) using namespace *ονομα_χωρου*;

Ολα τα μελη που οριζονται μεσα στο ***ονομα_χωρου*** μπορουν να χρησιμοποιηθουν χωρις τον τελεστη εμβελειας.
Παραδειγμα

```
using namespace mysp;  // Ολα τα μελη ειναι ορατα
x = 5;
myfn(7);
```

B) using *ονομα_χωρου*::*μελος_χωρου*;

Μονο ενα μελος του χωρου ειναι ορατο.
Παραδειγμα

```
using mysp::x;      // Μονο το μελος x ειναι ορατο
x = 2;
```

Τελεστης typeid

Μπορουμε να βρουμε τον τυπο ενος αντικειμενου κατα την διαρκεια εκτελεσης του προγραμματος με τον τελεστη **typeid**. Για να κανουμε χρηση του **typeid** πρεπει να συμπεριλαβουμε στο προγραμμα μας το αρχειο επικεφαλιδων **typeinfo.h**

```
#include <iostream.h>
#include <typeinfo.h>
int main()
{
    cout << typeid(345+22).name() << endl; // Παρασταση
    cout << typeid(127U).name() << endl;   // Σταθερα
    double d;
    cout << typeid(d).name() << endl;      // Μεταβλητη
```

```
    return(0);
}
```
Αποτελεσμα
```
int
unsigned int
double
```

ΕΙΣΟΔΟΣ/ΕΞΟΔΟΣ (Input/Output)

Η C++ υποστηριζει το συστημα εισοδου/εξοδου (Ε/Ε) της C. Επι πλεον οριζει δικο της αντικειμενοστρεφες συστημα Ε/Ε.

Ροη (**stream**) ειναι μια λογικη συσκευη που λαμβανει ή αποστελει πληροφοριες. Μια ροη συνδεεται με μια φυσικη συσκευη μεσω του συστηματος Ε/Ε της C++. Επειδη

ολες οι ροες συμπεριφερονται το ιδιο (ανεξαρτητως της φυσικης συσκευης που συνδεονται) για αυτον τον λογο οι συναρτησεις Ε/Ε λειτουργουν σε ολους του τυπους των φυσικων συσκευων. Ετσι, μπορει να χρησιμοποιηθει η ιδια συναρτηση για να τυπωσουμε κατι στον εκτυπωτη ή να εμφανισουμε κατι στην οθονη.

Καθε προγραμμα ανοιγει αυτοματα τις παρακατω ροες:

Ροη	**Συσκευη**	**Περιγραφη**
cin	**Πληκτρολογιο**	Καθιερωμενη Εισοδο
cout	**Οθονη**	Καθιερωμενη Εξοδο
cerr	**Οθονη**	Καθιερωμενη Εξοδο Λαθους
clog	**Οθονη**	Η cerr αλλα με χρηση buffer

Λειτουργικα συστηματα (DOS, Unix, OS/2, Windows, κ.ά) που υποστηριζουν αλλαγη κατευθυνσης Ε/Ε (I/O redirection) οι καθιερωμενες ροες μπορουν να αλλαξουν κατευθυνση σε αλλες συσκευες ή αρχεια.

Το συστημα Ε/Ε της C++ οριζεται στο αρχειο επικεφαλιδων (header file) **iostream.h** Σε αυτο το αρχειο οριζονται δυο ταξεις (classes) η **streambuf** (για απλές λειτουργιες Ε/Ε) και η **ios** (για μορφοποιημενη Ε/Ε - formated I/O) που υποστηριζουν τις λειτουργιες Ε/Ε. Απο την **ios** παραγονται (derived) οι classes **istream**, **ostream**, **iostream**.

Μορφοποιημενη Ε/Ε με μελη της ios

Με καθε ροη συνδεεται ενα συνολο απο "σημαιες" (flags) που ελεγχουν τον τροπο μορφοποιησης (format) της πληροφοριας. Οι σημαιες μορφοποιησης της ios ειναι ορισμενες σε ενα enum.

```
enum {                                  Οταν η σημαια ειναι "ενεργοποιημενη" 1 (set)
   skipws = 0x0001;          Τα προπορευομενα κενα απορριπτονται.
   left   = 0x0002;          Το γραψημο γινεται απο αριστερα προς δεξια.
   right  = 0x0004;          Το γραψημο γινεται απο δεξια προς αριστερα.
   internal=0x0008;
   dec    = 0x0010;          Εμφανιση στο δεκαδικο συστημα μετρησης.
   oct    = 0x0020;          Εμφανιση στο οκταδικο συστημα μετρησης.
   hex    = 0x0040;          Εμφανιση στο δεκαεξαδικο συστημα μετρησης.
   showbase=0x0080;          Εμφανιζει την βαση του συστηματος μετρησης.
   showpoint=0x0100;         Εμφανιζει την υποδιαστολη.
   uppercase=0x0200;         Οι χαρακτ x,e εμφανιζονται με κεφαλαια 0X1F, 0.1E-3
   showpos=0x0400;           Το + εμφανιζεται στους θετικους αριθμους.
   scientific=0x0800;        Επιστημονικη γραφη των πραγματικων αριθμων
   fixed = 0x1000;           Εξι δεκαδικα ψηφια ακριβειας εμφανιζονται
   unitbuf = 0x2000;         Το συστημα Ε/Ε "σωζεται" (flushed) μετα απο
};                           απο καθε λειτουργια εξοδου.
```

Για να ενεργοποιησουμε μια "σημαια" πρεπει να εκτελεσουμε την συναρτηση **setf()** που ειναι μελος της ios.

```
long setf(long σημαιες);
```

Για να απενεργοποιησουμε μια "σημαια" πρεπει να εκτελεσουμε την συναρτηση **unsetf()** που ειναι μελος της ios.

```
long unsetf(long σημαιες);
```

-- Παραδειγμα

```
#include <iostream.h>
int main()
{
    cout.setf(ios::hex);
    cout.setf(ios::showbase);
    cout << 100;
    return(0);
}
```

Αποτελεσμα
Θα εμφανισθει το **0x64**

Παραδειγματα:

```
α) cout.setf(ios::uppercase | ios::scientific);
   cout << 100.34 ;
```
Θα εμφανισθει **1.0034E+02**
```
   :
   cout.unsetf(ios::uppercase);
   cout << 100.34 ;
```
Θα εμφανισθει **1.0034e+02**

```
β) cout.setf(ios::showpos | ios::showpoint);
   cout << 70.00 ;
```
Θα εμφανισθει **+70.000000**

```
γ) cout.setf(ios::showbase | ios::hex);
   cout << 100;
```
Θα εμφανισθει **0x64**
```
   :
   cout.setf(ios::uppercase);
   cout << 100;
```
Θα εμφανισθει **0X64**

Στην ios υπαρχει η συναρτηση μελος **flags()** η οποια επιστρεφει την τρεχουσα κατασταση των "σημαίων".

```
long flags();
```
Παραδειγμα
```
long k;
:
k = cout.flags();
```

-- Να γραφει μια συναρτηση η οποια εμφανιζει την κατασταση μιας σημαιας.

```
#include <iostream.h>

void emfsimea(char *,long);

int main()
{
   long r;
   r = ios::right;      // Η τιμη για την σημαια right
   emfsimea("Right",r); // Εμφανισε την κατασταση της σημαιας
```

```
   cout.setf(ios::right); // "Αναψε" την σημαια
   emfsimea("Right",r); // Εμφανισε την κατασταση της σημαιας
   return(0);
}

void emfsimea(char *s, long x)
{
    long f;
    f = cout.flags();  // Διαβασε την κατασταση των σημαιων
    cout << "Η σημαια " << s << " ειναι ";
    if( f & x )
        cout << "ενεργοποιημενη (ON) " << endl;
    else
        cout << "απενεργοποιημενη (OFF) " << endl;
}
// p63.cpp //
```

Συναρτηση width

Καθοριζει τον χωρο στο οποιο θα εμφανισθει μια τιμη.

```
int width(int w);
```

Παραδειγμα: Βλεπε συναρτηση fill().

Συναρτηση precision

Καθοριζει τα ψηφια ακριβειας για την εμφανιση ενος πραγματικου. Εξ' ορισμου ειναι 6 ψηφια. Το δεκαδικο σημειο δεν μετραται.

```
int precision(int p);
```

Παραδειγματα

```
α) cout << 123456.789; // Θα εμφανισθει 123456
β) cout << 12345.6789; // Θα εμφανισθει 12345.6
γ) cout.precision(4);
   cout << 12.345;     // Θα εμφανισθει 12.34
δ) cout.precision(5);
   cout << 1234.56789  // Θα εμφανισθει 1234.5
```

Συναρτηση fill

Καθοριζει τον χαρακτηρα που θα "γεμισει" ενα πεδιο (εξ' ορισμου ειναι το κενο διαστημα).

```
int fill(char ch);
```

Παραδειγμα

```
#include <iostream.h>
main()
{
    cout.precision(4);
    cout.fill('*');
    cout.width(10);
    cout << 12.34567;  // Θα εμφανισθει *****12.34
    cout << "Adam";    //     ...       ******Adam
    cout.setf(ios::left);
    cout << 12.34567;  // Θα εμφανισθει 12.34*****
    return(0);
}
```

ΑΡΧΕΙΑ Ε/Ε (File I/O)

Τα αρχεια ειναι μια ειδικη περιπτωση του συστηματος Ε/Ε. Για να χρησιμοποιειθει ενα αρχειο πρεπει να συνδεθει με μια ροη (stream).

Υπαρχουν τρεις τυποι ροης: **εισοδου** (input), **εξοδου** (output), **εισοδου/εξοδου** (input/output).

Για να δημιουργηθει μια ροη πρεπει να δηλωθει ως μια απο τις ακολουθες ταξεις: **`ifstream`** (για ροη εισοδου), **`ofstream`** (για ροη εξοδου), **`fstream`** (για ροη εισοδου/εξοδου). Οι ταξεις αυτες οριζονται στο αρχειο επικεφαλιδων **`fstream.h`** Οι ταξεις ifstream, ofstream, fstream παραγονται (derived) απο **`istream`**, **`ostream`** (οι οποιες με την σειρα τους παραγονται απο την ios).

Υπαρχουν δυο τυποι αρχειων:
1) Κειμενου (text - ascii)
2) Δυαδικου (binary).

Δεικτες Θεσης

Το συστημα Ε/Ε της C++ διαχειριζεται **δυο δεικτες-θεσης** (δεικτης-θεσης ειναι ενας ακεραιος αριθμος - long int) για καθε αρχειο:
α) ο δεικτης-θεσης-διαβασματος (get pointer), καθοριζει την θεση (σε πιο byte) στο αρχειο οπου θα γινει η επομενη λειτουργια εισοδου,
β) ο δεικτης-θεσης-γραψηματος (put pointer), καθοριζει την θεση (σε πιο byte) στο αρχειο οπου θα γινει η επομενη λειτουργια εξοδου.
Οταν ενα αρχειο ανοιγει (συναρτηση **`open`**) τοτε οι δεικτες-θεσης τοποθετουνται στο πρωτο byte στην αρχη του αρχειου (η αριθμηση αρχιζει απο το μηδεν).
Καθε φορα που εκτελειται μια συναρτηση διαβασματος (οπως **`read`**, **`get`**, κλπ) ο δεικτης-θεσης-διαβασματος μετακινειται αυτοματα στην θεση εκει που θα γινει το επομενο διαβασμα.
Επισης καθε φορα που εκτελειται μια συναρτηση γραψηματος (οπως **`write`**, **`put`**, κλπ) ο δεικτης-θεσης-γραψηματος μετακινειται αυτοματα στην θεση εκει που θα γινει το επομενο γραψημο.
Με τις συναρτησεις **`tellg`** και **`tellp`** μπορομε να βρουμε τις θεσεις των δυο δεικτων-θεσης.
Με τις συναρτησεις **`seekg`** και **`seekp`** μπορουμε να μετακινισουμε και να τοποθετησουμε τους δεικτες-θεσεις σε οποιοδηποτε σημειο στο αρχειο θελουμε.

Τελος Αρχειου (end of file)

Καθε αρχειο εχει μια ετικετα που σημειωνει το τελος του αρχειου (end of file). Η ετικετα ειναι ενας χαρακτηρας. Η συναρτηση **`eof`** ελεγχει για το εαν ενας δεικτης-θεσης ειναι στο τελος-αρχειου. (Σε αρχεια κειμενου στο DOS ειναι ο χαρακτηρας 26 δηλ. ctrl-Z, στο UNIX ειναι ο χαρακτηρας 4 δηλ. ctrl-D)

Buffer

Τα δεδομενα που γραφονται σε ροη ή διαβαζονται απο μια ροη συλλεγονται σε ενα ενδιαμμεσο χωρο που ονομαζεται "buffer" Σε εργασιες "γραφής" (write) το περιεχομενο του buffer εξοδου γραφεται στη καταλληλη τελικη τοποθεσια (φυσικη συσκευη) οταν: **α**) το buffer γεμισει, **β**) οταν κλεισει η ροη, **γ**) οταν το προγραμμα

τελειωνει χωρις προβλημα **δ**) οταν εκτελεσθει η συναρτηση **flush** (σε ορισμενες γλωσσες η εντολη ονομαζεται commit). Οταν αυτο συμβαινει λεμε οτι ο buffer "flushed". Οταν το προγραμμα τελειωνει με προβλημα τοτε ο buffer εξοδου πιθανον να μη κατεβασει (flush) και σωσει τα δεδομενα με αποτελεσμα να χαθουν δεδομενα.

Συναρτησεις E /E

open

Εφ' οσον δημιουργηθει μια ροη μπορει να συνδεθει με ενα αρχειο χρησιμοποιοντας την συναρτηση open(). Η συναρτηση open() ανοιγει ενα αρχειο, εαν δεν υπαρχει το δημιουργει.

```
void open(const char *ονομα_αρχειου, int εργασια);
```

Οπου *εργασια*	Περιγραφη
ios::app	- Επεκταση στο τελος του αρχειου.
ios::ate	- Ο δεικτης θεσης του αρχειου τοποθετειται στο τελος οταν το αρχειο ανοιγει.
ios::binary	- Αρχειο binary.
ios::in	- Αρχειο εισοδου.
ios::nocreate	- Να μη δημιουργηθει, εαν δεν υπαρχει το αρχειο.
ios::noreplace	- Να μη γινει αντικατασταθει, εαν υπαρχει το αρχειο.
ios::out	- Αρχειο εξοδου.
ios::trunc	- Το παλαιο περιεχομενο του αρχειου καταστρεφεται.

Παραδειγματα

```
α) ofstream os;  // Ροη εξοδου os
   :
   os.open("test");
```

ιδιο με:

```
   ofstream os;  // Ροη εξοδου os
   :
   os.open("test", ios::out);
```

```
β) fstream mys;
   :
   mys.open("data", ios::in | ios::out);
```

Εαν υπαρχει προβλημα στο open τοτε η ροη mys ειναι μηδεν.

```
   if (!mys) {
       cout << "Προβληµα στο ανοιγµα του αρχειου.\n";
       :
   }
```

get και put

Οι συναρτησεις get() και put() χειριζονται δεδομενα σε επιπεδο χαρακτηρα (byte).

```
istream &get(char &ch);
ostream &put(char ch);
```

Η **get** διαβαζει εναν χαρακτηρα *ch* απο μια ροη. Η **put** γραφει (τοποθετει) εναν χαρακτηρα *ch* σε μια ροη. Εαν τα δεδομενα στην ροη ειναι δυαδικα (binary) τοτε η ροη πρεπει να ανοιχθει με ios::binary.

read και write

Με τις συναρτησεις read() και write() μπορουμε να γραψουμε ή να διαβασουμε κοματια μνημης (blocks) απο δεδομενα.

```
istream &read(unsigned char *buf, int num);
ostream &write(const unsigned char *buf, int num);
```

Η συναρτηση **read** διαβαζει *num* χαρακτηρες (bytes) στο χωρο που δειχνει ο δεικτης *buf*. Η συναρτηση **write** γραφει *num* χαρακτηρες απο τον χωρο που δειχνει ο δεικτης *buf*. Η συναρτηση **write** δεν σταματα οταν συναρτηση εναν τερματικο (null) χαρακτηρα.

-- Παραδειγμα. Δημιουργει μια ροη-αρχειου και γραφει την δυαδικη τιμη της δομης Date.

```
#include <fstream.h>
struct Date
{
   int mo, da, yr;
};

int main()
{
    Date dt = { 20, 7, 97 };
    ofstream tfile( "date.dat" , ios::binary );
    tfile.write( (char *) &dt, sizeof(dt));
    return(0);
}
```

get

Εκτος απο την προηγουμενη περιγραφη της, η συναρτηση get ειναι υπερφορτωμενη με διαφορους τροπους.

```
istream &get(char *buf, int num,char delim='\n');
```

Διαβαζει *num* χαρακτηρες στο χωρο που δειχνει ο *buf*, εκτος εαν πιο νωρις (πριν συμληρωση το αριθμο *num*) συναντηση το χαρακτηρα που οριζεται ως *delim*. Εαν ο χαρακτηρας *delim* δεν ορισθει τοτε εξ'ορισμου ειναι ο χαρακτηρας '\n' (new line).

getline

Διαβαζει *num* χαρακτηρες στο χωρο που δειχνει ο *buf*, εκτος εαν πιο νωρις (πριν συμληρωση το αριθμο *num*) συναντηση το χαρακτηρα που οριζεται ως *delim*. Εαν ο χαρακτηρας *delim* δεν ορισθει τοτε εξ'ορισμου ειναι ο χαρακτηρας '\n'. Ειναι ισοδυναμη με την get(char *buf, int num,char delim) με μονη διαφορα οτι η **getline** διαγραφει τον χαρακτηρα delim '\n' στον *buf*.

```
istream &getline(char *buf, int num,char delim='\n');
```

ignore

Διαβαζει και αγνοει *num* χαρακτηρες (1 εξ'ορισμου) εκτος και αν πιο νωρις συναντησει τον χαρακτηρα *delim* (EOF end of file - εξ'ορισμου χαρακτηρας τελους αρχειου) απο μια ροη εισοδου.

```
istream &ignore(int num=1, int delim=EOF);
```

Παραδειγμα:

```
ifstream si;
:
si.ignore(10, ' '); // Αγνοησε 10 χαρακτηρες εκτος και εαν
                    // συναντησεις πιο μπροστα το κενο διαστημα
```

peek

Η συναρτηση peek μας διδει τον επομενο χαρακτηρα απο την ροη χωρις ομως να μετακινηθει ο δεικτης θεσης στην ροη.

```
int peek();
```

Παραδειγμα: Βλεπε συναρτηση putback()

putback

Η συναρτηση putback() τοποθετει πισω τον τελευταιο χαρακτηρα που διαβαστηκε σε μια ροη.

```
istream &putback(char c);
```

-- Το παρακατω προγραμμα διαβαζει μια σειρα χαρακτηρες απο την καθιερωμενη ροη **cin** εαν ο χαρακτηρας ειναι "πεζος" μετατρεπεται σε κεφαλαιο και τοποθετειται πισω στην ροη.

```
#include <iostream.h>
int main()
{
    char ch;
    do {
       ch = cin.peek();// Φερε τον χαρακτηρα αλλα μη προχωρας
                       // τον δεικτη-διαβασματος.
       if(ch >= 'a' && ch <= 'z') {
           ch = cin.get();  // Διαβασε τον "πεζο" χαρακτηρα.
           ch = ch - 32;    // Μετατροπη σε κεφαλαιο.
           cin.putback(ch); // Τοποθετησε τον χαρακτηρα πισω
       }                    // στην ροη

       ch = cin.get();
       cout << ch;       // Εμφανισε τον χαρακτηρα στην οθονη
    } while (ch != '\n');
    return(0);
}
// p43.cpp //
```

flush

Η συναρτηση flush "σωζει" στον δισκο ολες τις πληροφοριες που βρισκονται στην προσωρινη μνημη (buffer).

```
osfream &flush();
```

Παραδειγμα

```
#include <fstream.h>
int main()
{
    ofstream so("test.dat");
    :
    so.flush();
    :
}
```

seekg και seekp

Η **seekg** επιτρεπει να μετακινησουμε τον δεικτη-θεσης-διαβασματος (get δεικτη) του αρχειου σε ενα καθορισμενο σημειο του αρχειου για "read" εργασιες.
Η **seekp** επιτρεπει να μετακινησουμε τον δεικτη-θεσης γραψηματος (put δεικτη) του αρχειου σε ενα καθορισμενο σημειο του αρχειου για "write" εργασιες.

```
istream &seekg(offset, start_position);
ostream &seekp(offset, start_position);
```

Το *offset* οριζει το πληθος των χαρακτηρων (bytes) που πρεπει να μετακινιθει ο δεικτης-θεσης παντα ως προς το σημειο αναφορας (*start_position*)
Η *start_position* οριζει την θεση του δεικτη-θεσης. Οι τιμες που μπορει να λαβει ειναι οι κατωθι:

Τιμη	Περιγραφη
ios::beg	Αρχη του αρχειου
ios::cur	Τρεχουσα θεση του αρχειου
ios::end	Τελος του αρχειου

Παραδειγμα: Βλεπε συναρτηση tellp()

tellg και tellp

Η συναρτηση **tellg** επιστρεφει την τρεχουσα θεση του δεικτη-διαβασματος.
Η συναρτηση **tellp** επιστρεφει την τρεχουσα θεση του δεικτη-γραψηματος.

```
θεση_στη_ροη tellg();
θεση_στη_ροη tellp();
```

Παραδειγμα

```
#include <iostream.h>
#include <fstream.h>
int main()
{
    ofstream fo;
    fo.open("rand.dat");
    if (!fo) {
        cout << "Λαθος κατα το ανοιγμα του αρχειου\n";
        return(1);
    }
    // Τοποθετηση του δεικτη-θεσης-γραψηματος στο τελος του
    // αρχειου
    fo.seekp(0, ios::end);
    // Επεξεργασια ...
    cout << fo.tellp();  // Εμφανιση του δεικτη-γραψηματος
    // Επεξεργασια ...
    fo.close();
    return(0);
```

```
}
// p42.cpp //
```

eof()

Η συναρτηση eof() αναγνωριζει και επιστρεφει μια μη μηδενικη τιμη οταν ο δεικτης θεσης αρχειου φθανει στο τελος του αρχειου.

```
int eof();
```

Ελεγχος επεξεργασιας

Το συστημα Ε/Ε της C++ διατηρει πληροφοριες για το αποτελεσμα καθε λειτουργιας Ε/Ε (I/O status). Υπαρχουν δυο τροποι για να λαβουμε πληροφοριες για την κατασταση Ε/Ε:
Α) με τις συναρτησεις ελεγχου.
Β) με την συναρτηση rdstate

Α) Συναρτησεις Ελεγχου

Οι συναρτησεις ειναι "συναρτησεις μέλη" που ελεγχουν τυχον λαθη κατα το γραψημο/διαβασμα σε μια ροη.

Συναρτηση	**Περιγραφη**
bad	Επιστρεφει μη-μηδενικη τιμη εαν υπαρχει ενα σοβαρο λαθος.
fail	Επιστρεφει μη-μηδενικη τιμη εαν υπαρχει ενα σοβαρο λαθος ή το αρχειο δεν υπαρχει. Η επεξεργασια μπορει να συνεχισθει μετα απο κληση της clear(0).
good	Επιστρεφει μη-μηδενικη τιμη εαν δεν υπαρχει λαθος και δεν εχει φθασει στο τελος του αρχειου (end-of-file not set).
eof	Επιστρεφει μη-μηδενικη τιμη (on end-of-file) οταν προσπαθησουμε να διαβασουμε περα απο την ετικετα τελος-αρχειου.
clear	Καθαριζει την κατασταση εσωτερικου λαθους (εξ' ορισμου καθαριζει ολα τα error bits).

Ο τελεστης **!** υπερφορτωνεται και εκτελει την ιδια λειτουργια οπως η συναρτηση **fail**, ετσι η εντολη

```
if( !cout ) ...
```

ειναι ιδια με

```
if( cout.fail() ) ...
```

Ο τελεστης **void*()** υπερφορτωνεται για να ειναι το αντιθετο του τελεστη **!**, ετσι η εντολη

```
if( cout ) ...
```

ειναι ιδια με

```
if( !cout.fail() ) ...
```

Ο τελεστης **void*()** δεν ειναι ισοδυναμος με την **good** επειδη δεν ελεγχει για το τελος-αρχειου.

```
if( cout ) ...
```

δεν ειναι ιδια με

```
if( cout.good() ) ...
```

Β) Συναρτηση rdstate

Η συναρτηση **rdstate** επιστρεφει την τρεχουσα κατασταση λαθων (current error state). Η τρεχουσα κατασταση κρατειται σε εναν ακεραιο στον οποιον κωδικοποιουνται οι παρακατω flags:

eofbit	1 = τελος-αρχειου
	Διαφορετικα 0
failbit	1 = οχι σοβαρο λαθος
	Διαφορετικα 0
badbit	1 = σοβαρο λαθος
	Διαφορετικα 0

Οι σημαιες (flags) αυτες βρισκονται στη **class ios**. Στην **ios** οριζεται και το **goodbit** που εχει την τιμη 0.

-- Παραδειγμα. Το προγραμμα διαβαζει ενα αρχειο χαρακτηρα-χαρακτηρα και το εμφανιζει στην οθονη. Για καθε χαρακτηρα που διαβαζεται γινεται ελεγχος για τυχον λαθη (τελος αρχειου, οχι σοβαρο λαθος, σοβαρο λαθος). Εαν προκυψει ενα απο τα λαθη θα εμφανισθει στην οθονη αναλογο μηνυμα.

```
#include <iostream.h>
#include <fstream.h>
int main(int argc, char *argv[])
{
    ifstream eis(argv[1]);  //  Ροη εισοδου
    if(!eis) {
       cout << "Προβλημα στο ανοιγμα του αρχειου εισοδου.\n";
       return(1);
    }
    char ch;
    while (eis.get(ch)) {
       cout << ch;
       elegxos(eis);
    }
    eis.close();
    return(0);
}

void elegxos(ifstream &eis)
{
    int i;
    i = eis.rdstate();
    if( i & ios::eofbit )
        cout << "Ετικετα Τ.Α. (τελος αρχειου εβρεθη) \n";
    esle if( i & ios::failbit )
        cout << "Οχι σοβαρο λαθος Ε/Ε \n";
    esle if( i & ios::badbit )
        cout << "Σοβαρο λαθος Ε/Ε \n";
}
```

Εισοδος/Εξοδος σε πινακες

Το συστημα Ε/Ε της C++ επιτρεπει λειτουργιες Ε/Ε σε πινακες (array I/O). Οι λειτουργιες αυτες εκτελουνται με τις συνηθισμενες ροες της C++. (Η Ε/Ε σε πινακες ειναι ιδια με την λειτουργια των συναρτησεων sscanf() και sprintf() της C)
Οι βασικες δομες, classes και σταθερες βρισκονται δηλωμενες στο **strstrea.h**

Οι ταξεις που χρησιμοποιουνται για Ε/Ε σε πινακες ειναι: **istrstream** (για εισοδο), **ostrstream** (για εξοδο), και **strstream** (για εισοδο/εξοδο) Οι ταξεις αυτες προερχονται απο την ταξη strstreambuf. Εκτος απο την strstreambuf, η istrstream προερχεται και απο την istream, η ostrstream προερχεται και απο την ostream, η δε strstream περιλαμβανει τις iostream ταξεις. Ετσι ολες οι ταξεις για Ε/Ε σε πινακες εχουν προσβαση στις ιδιες "συναρτησεις μελη" που εχουν οι "συνηθισμενες" ταξεις Ε/Ε.

Για να συνδεσουμε μια ροη εξοδου με ενα πινακα χρησιμοποιουμε το:
```
ostrstream onoma(char *buf,int mikos,int ergasia = ios::out);
```

Για να συνδεσουμε μια ροη εισοδου με ενα πινακα χρησιμοποιουμε το:
```
istrstream onoma_rois_eis(char *buf);
```
ή
```
istrstream onoma_rois_eis(char *buf,int mikos);
```
Μονο το κομματι ισον με το *mikos* θα χρησιμοποιηθει ως εισοδο.

Για να συνδεσουμε μια ροη εισοδου/εξοδου με ενα πινακα χρησιμοποιουμε το:
```
strstream r_eis_ex(char *buf, int mikos, int ergasia);
```
Παραδειγμα:
```
char b[80];
strstream ee(b, sizeof(b), ios::in | ios::out);
```

Συναρτηση pcount()
Επιστρεφει το μηκος μιας ροης εξοδου σε πινακα. Στο μηκος συμπεριλαμβανεται και ο τερματικος χαρακτηρας.
```
int pcount();
```

-- Παραδειγμα ροης εξοδου σε πινακα.

```
#include <iostream.h>
#include <strstrea.h>
int main()
{
   char s[80];
   ostrstream exs(s, sizeof(s));// Ροη εξοδου exs σε πινακα s
   exs << "Mirror ";
   exs << '7';
   exs << " " << 100 - 3*5 << hex << " ";
   exs.setf(ios::showbase);
   exs << 126 << " " << 123.4 << ends;

   cout << s << endl;   // Εμφανηση του πινακα στην οθονη
   cout << "Μηκος ροης " << exs.pcount() << endl;

   return(0);
}
// p66.cpp //
```
Αποτελεσμα

```
Mirror 7 85 0x7e 123.4
Μηκος ροης 23
```

-- Παραδειγμα Ροης εισοδου σε πινακα.

```
#include <iostream.h>
#include <strstrea.h>
int main()
{
    char b[] = "20 Rainmaker 12.34 C";
    istrstream eis(b);  // Ροη εισοδου eis σε πινακα s
    int i;
    char s[80];
    eis >> i;   // Διαβαζει το: 20
    eis >> s;   // Διαβαζει το: Rainmaker
    cout << i << " " << s << endl;  // Εμφανιση

    float f;
    char ch;
    eis >> f;                       // Διαβαζει το: 12.34
    eis >> ch;                       // Διαβαζει το: C
    cout << f << " " << ch << endl;  // Εμφανιση
    return(0);
}
// p68.cpp //
```

Αποτελεσμα

```
20 Rainmaker
12.34 C
```

-- Παραδειγμα Ροης εισοδου/εξοδου σε πινακα.

```
#include <iostream.h>
#include <strstrea.h>
int main()
{
    char bee[80];
    strstream ees(bee, sizeof(bee), ios::in | ios::out);
    ees << "10 20 Friends";  // Εισοδος

    int x,y; char s[80];
    ees >> x >> y >> s;   // Εξοδος
    cout << x << " " << y << " " << s << endl; // Εμφανιση
    ees.seekg(7, ios::beg); // Τυχαια αναζητηση στη ροη ees
    char ch;
    ees >> ch;
    cout << "Ο χαρ. στην θεση 7 ειναι " << ch << endl;
    return(0);
}
// p69.cpp //
```

-- Οι συναρτησεις μελη οπως get, put, eof κλπ μπορουν να χρησιμοποιηθουν και σε ροες-πινακες. Παραδειγμα

```
#include <iostream.h>

#include <strstrea.h>
int main()
```

```
{
    char b[] = "Photographic memories";
    istrstream es(b);  // Ροη εσοδου es σε πινακα s
    char ch;
    while(es.get(ch))  // Διαβασε χαρακτηρα
        cout << ch;    // Εμφανισε τον
    return(0);
}
// p67.cpp //
```

-- Παραδειγμα τυχαιας αναζητησης ροης εισοδου/εξοδου σε πινακα

```
#include <iostream.h>
#include <strstrea.h>
int main()
{
    char bee[80];
    strstream ees(bee, sizeof(bee), ios::in | ios::out);
    ees << "10 20 Friends";  // Εισοδος
    ees.seekg(6, ios::beg); // Τυχαια αναζητηση στη ροη ees

    char ch;
    ees >> ch;
    cout << "Ο χαρ. στην θεση 6 ειναι " << ch << endl;

    return(0);
}
```

Αποτελεσμα

```
Ο χαρ. στην θεση 6 ειναι F
```

Σχολιο: Ο δεικτης στην seekg αρχιζει απο την θεση 0 (μηδεν)

Ασκησεις

-- Να γραφει ενα προγραμμα το οποιο δημιουργει ενα αρχειο κειμενου. Σε καθε γραμμη καταχωρουνται στοιχεια πελατων το ονομα και η χρεωση. Τα δεδομενα διδονται απο το πληκτρολογιο. Τελος εισαγωγης δεδομενων με χρεωση = 0.

```
#include <iostream.h>
#include <fstream.h>
int main()
{
    ofstream rex;  //  Ροη εξοδου
    rex.open("test");
    if(!rex) {
        cout << "Προβλημα στο ανοιγμα του αρχειου.\n";
        return(1);
    }
    char nam[20];  float xr;
    while (1) {
       cout << "Ονομα: " ;
       cin >> nam;
       cout << "Χρεωση: " ;
       cin >> xr;
       if(xr == 0)
           break;
       rex << nam << " " << xr << endl;  // Οπως η fprintf
    }                                    // της C.
    rex.close();
    return(0);
}
// p39.cpp //
```

-- Να γραφει ενα προγραμμα το οποιο εμφανιζει ενα αρχειο κειμενου. Σε καθε γραμμη υπαρχουν καταχωρημενα στοιχεια πελατων το ονομα και η χρεωση.

```
#include <iostream.h>
#include <fstream.h>
int main()
{
    ifstream reis("test");  //  Ροη εισοδου
    if(!reis) {
        cout << "Προβλημα στο ανοιγμα του αρχειου.\n";
        return(1);
    }
    char nam[20];  float xr;
    while (1) {
        reis >> nam >> xr;   // Οπως η fscanf της C.
        if(!reis)     // Ιδιο με: if(reis.eof())
            break;
        cout << "Ονομα: " << nam << "  " << "Χρεωση: " << xr
             << endl;
    }
    reis.close();
    return(0);
}
// p40.cpp
```

-- Να γραφει ενα προγραμμα το οποιο εμφανιζει ενα αρχειο κειμενου. Σε καθε γραμμη υπαρχουν καταχωρημενα στοιχεια πελατων το ονομα και η χρεωση. Το διαβασμα θα γινει χαρακτηρα-χαρακτηρα.

```
#include <iostream.h>
#include <fstream.h>
int main()
{
    ifstream rin("test");  //  Ροη εισοδου
    if(!rin) {
        cout << "Προβλημα στο ανοιγμα του αρχειου.\n";
        return(1);
    }
    char ch;
    while (1) {
        rin.get(ch);   // Οπως η fgetc της C.
        if(rin.eof())
            break;
        cout << ch;
    }
    rin.close();
    return(0);
}
// p41.cpp //
```

Σχολιο. Η παραπανω θηλεια while { } μπορει να γραφει και ως:

```
    while (rin.get(ch))
        cout << ch;
```

-- Να γραφει ενα προγραμμα το οποιο αντιγραφει το αρχειο ITEM.DAT σε ενα αλλο αρχειο το ονομα του οποιου διδεται απο την γραμμη των εντολων. (Το διαβασμα και το γραψημο γινεται χαρακτηρα-χαρακτηρα).

```
#include <iostream.h>
#include <fstream.h>
int main(int argc, char *argv[])
{
    ifstream es("item.dat");  //  Ροη εισοδου
    ofstream ex(argv[1]);     //  Ροη εξοδου
    if(!es) {
       cout << "Προβλημα στο ανοιγμα του αρχειου εισοδου.\n";
       return(1);
    }
    if(!ex) {
       cout << "Προβλημα στο ανοιγμα του αρχειου εξοδου.\n";
       return(2);
    }
    char ch;
    while (1) {
       es.get(ch);
       if(es.eof())
          break;
       ex.put(ch);
    }

    es.close();
    ex.close();
```

```
    return(0);
}
// p44.cpp //
```

Σχολιο. Η παραπανω θηλεια while { } μπορει να γραφει και ως:

```
    while(es.get(ch))
       ex.put(ch);
```

-- Να γραφει ενα προγραμμα το οποιο αντιγραφει το περιεχομενο ενος αρχειου κειμενου σε ενα αλλο αρχειο. Τα ονοματα των αρχειων διδονται απο την γραμμη των εντολων. (Το διαβασμα και το γραψημο γινεται γραμμη-γραμμη).

```
#include <iostream.h>
#include <fstream.h>
int main(int argc, char *argv[])
{
    ifstream es(argv[1]);   // Ροη εισοδου
    ofstream ex(argv[2]);   // Ροη εξοδου
    if(!es) {
       cout << "Προβλημα στο ανοιγμα του αρχειου εισοδου.\n";
       return(1);
    }
    if(!ex) {
       cout << "Προβλημα στο ανοιγμα του αρχειου εξοδου.\n";
       return(2);
    }
    char buf[100];
    while (1) {
       es.getline(buf,sizeof(buf),'\n');// Διαβασε μια γραμμη
                                        // στο buf.
       if(es.eof())        // Τελος αρειου ?
          break;
       ex << buf << '\n';  // Γραψε τον buf+'\n'
    }
    es.close();
    ex.close();
    return(0);
}
// p45.cpp //
```

-- Να γραφει ενα προγραμμα το οποιο δημιουργει ενα αρχειο εγγραφων. Οι εγγραφες περιεχουν στοιχεια πελατων (κωδικος, επωνυμο(ια), τηλεφωνο, ποσο χρεωσης, ποσο πιστωσης) μιας εταιρειας.

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <string.h>

// Ορισμος εγγραφης
struct UC01 {
    int kod;
    char eponym[30];
    char tel[15];
    long xreos,pist;
};
typedef struct UC01 PEL_REC;
```

```
// Πρωτοτυπα συναρτησεων
int  rd(PEL_REC &);
void sav(PEL_REC , ofstream &);

int main()
{
    PEL_REC rec1;
    ofstream so;
    int y;
    so.open("PELFIL",ios::binary);
    if (so.fail()) {
        cout << "Λαθος ονομα αρχειου ή προβλημα στο αρχειο"
             << endl;
        return(1);
    }
    while (1) {
        y = rd(rec1); // Διαβασε τα δεδομενα απο το τερματικο
        if (y == 0)
            break;
        sav(rec1,so); // Γραψε τα δεδομενα στον δισκο
    }
    return(0);
}

// Η συναρτηση sav γραφει μια εγγραφη στο αρχειο
// Επιστρεφει: void
void sav(PEL_REC rec, ofstream &so)
{
    so.write((char *) &rec,sizeof(PEL_REC));
}

// Η συναρτηση rd διαβαζει απο το τερματικο δεδομενα και τα
// αποθηκευει στην δομη (εγγραφη)
// Επιστρεφει: 1 η καταχωρηση συνεχιζεται
//             0 τελος καταχωρησης
int rd(PEL_REC &rec)
{
    char bf[256];
    cout << "ΕΙΣΑΓΩΓΗ ΣΤΟΙΧΕΙΩΝ ΠΕΛΑΤΗ" << endl;
    cout << " Κωδικος: " ; cin >> bf;  rec.kod = atoi(bf);
    if (rec.kod == 0)
        return(0);
    cout << " Επωνυμο: ";  cin >> bf;  strcpy(rec.eponym,bf);
    cout << "Τηλεφωνο: ";  cin >> bf;  strcpy(rec.tel,bf);
    cout << "  Χρεωση: ";  cin >> bf;  rec.xreos = atol(bf);
    cout << " Πιστωση: ";  cin >> bf;  rec.pist = atol(bf);
    return(1);
}
// p32a.cpp //
```

-- Να γραφει ενα προγραμμα το οποιο εμφανιζει ολα τα στοιχεια ολων των εγγραφων του αρχειου πελατων που δημιουργηθηκε με το προηγουμενο προγραμμα.

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
```

```
// Ορισμος εγγραφης
struct UC01 {
    int kod;
    char eponym[30];
    char tel[15];
    long xreos,pist;
} rec1;

int rr(ifstream &); // Πρωτοπυπα συναρτησεων
void dsp();

int main()
{
    ifstream si("PELFIL");  // Ροη εισοδου
    if (!si) {
        cout << "Λαθος ονομα αρχειου ή προβλημα στο αρχειο"
             << endl;
        return(1);
    }

    int y;
    while (1) {
        y = rr(si);  // Διαβασε μια εγγραφη
        if (y == 0)  // Τελος εγγραφών ?
            break;
        dsp();       // Εμφανισε την εγγραφη
    }
    return(0);
}

// Επιστρεφει: 0 = τελος του αρχειου
//             Διαφορετικα 1.
int rr(ifstream &si)  // Η συναρτηση rr διαβαζει μια εγγραφη
{
    int i = 1;
    si.read((char*) &rec1,sizeof(rec1));
    if(si.eof())
        i = 0;
    return(i);
}

// Η συναρτηση dsp εμφανιζει μια εγγραφη.
void dsp()
{
    cout << "ΕΜΦΑΝΙΣΗ ΣΤΟΙΧΕΙΩΝ ΠΕΛΑΤΗ" << endl;
    cout << " Κωδικος: " << setw(6) << rec1.kod
         << endl;
    cout << " Επωνυμο: " << rec1.eponym << endl;
    cout << "Τηλεφωνο: " << rec1.tel    << endl;
    cout << "  Χρεωση: " << setw(6) << rec1.xreos
         << endl;
    cout << " Πιστωση: " << setw(6) << rec1.pist
         << endl;
}
// p33a.cpp //
```