



**Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών**

Εισαγωγή στον Προγραμματισμό

Η γλώσσα προγραμματισμού C

Τελεστές και Τελεστέοι

Τελεστές και Τελεστέοι

□ Τελεστής (operator)

- Σύμβολο που αναπαριστά μία συγκεκριμένη στοιχειώδη διεργασία η οποία ενεργεί πάνω σε κάποια δεδομένα και παράγει κάποιο αποτέλεσμα
- Κοινώς: μια «πράξη» (+, -, *, /)
- Αλλά και μια σύγκριση (<, <=, ==, >=, >), μια ανάθεση (=)
- Και πολλά άλλα

□ Τελεστέος (operand)

- Τα δεδομένα (σταθερές, μεταβλητές, κλήσεις συναρτήσεων)
- Π.χ., το **α** και το **β** στην έκφραση **α+β**

Κατηγορίες Τελεστών

- Αριθμητικοί
 - +, -, *, /, %
- Λογικοί
 - &&, ||, !
- Σύγκρισης ή Συσχετιστικοί
 - >, >=, ==, !=, <, <=
- Διαχείρισης ψηφίων
 - >>, <<, &, |, ^, ~
- Διαχείρισης μνήμης
 - &, [], ., ->
- Ειδικοί
 - ++, --, +=, -=, *=, /=, %=, &=, |=, <<=, >>=

Τελεστές διαχείρισης ψηφίων

- `&` AND ανά bit
 - `|` OR ανά bit
 - `^` αποκλειστικό OR ανά bit
 - `<<` ολίσθηση αριστερά ανά bit
 - `>>` ολίσθηση δεξιά ανά bit
 - `~` συμπλήρωμα ως προς 1 (δηλ, αντιστροφή ανά bit)
-
- Ο τελεστής AND (`&`) χρησιμοποιείται και για εφαρμογή μάσκας σε ένα σύνολο από bits, ενώ ο τελεστής OR (`|`) για να θέσει ένα σύνολο bits σε 1. Π.χ.:
 - η εντολή `k=k & 7` θέτει σε 0 όλα τα bit του `k` εκτός από τα 3 δεξιότερα, τα οποία αφήνει ανέπαφα
 - η εντολή `k=k | 7` θέτει τα 3 δεξιότερα bit του `k` σε 1, αφήνοντας ανέπαφα όλα τα υπόλοιπα

Παράδειγμα

```
#include <stdio.h>

main() {
    unsigned int a = 60; /* 60 = 00000000 00000000 00000000 00111100 */
    unsigned int b = 13; /* 13 = 00000000 00000000 00000000 00001101 */
    int c = 0;
    c = a & b; /* 12 = 00000000 00000000 00000000 00001100 */
    printf("c = a & b = %d\n", c );
    c = a | b; /* 61 = 00000000 00000000 00000000 00111101 */
    printf("c = a | b = %d\n", c );
    c = a ^ b; /* 49 = 00000000 00000000 00000000 00110001 */
    printf("c = a ^ b = %d\n", c );
    c = ~a; /* -61 = 11111111 11111111 11111111 11000011 */
    printf("c = ~a = %d\n", c );
    c = a << 2; /* 240 = 00000000 00000000 00000000 11110000 */
    printf("c = a << 2 = %d\n", c );
    c = a >> 2; /* 15 = 00000000 00000000 00000000 00001111 */
    printf("c = a >> 2 = %d\n", c );
}
```

Ειδικοί Τελεστές

- Μοναδιαίοι τελεστές αύξησης/μείωσης (++ , --)
 - **++x** αυξάνει το x κατά ένα, και επιστρέφει την αυξημένη τιμή
 - **x++** αυξάνει το x κατά ένα, και επιστρέφει την τιμή προ της αύξησης

- Σύνθετοι τελεστές ανάθεσης (+=, -=, *=, /=)
 - **x += 10;** ισοδύναμο με το **x = x + 10;**
 - **x *= y+1;** ισοδύναμο με το **x = x*(y+1);**
 - **x <= 2;** ισοδύναμο με το **x = x < 2;** (αριστ. ολίσθ. κατά 2 ψηφία)

- Τριαδικός τελεστής (ternary operator) (? :)
 - <expr1> ? <expr2> : <expr3>
 - **a > b ? a : b** “Αν a>b, τότε a, αλλιώς b”, κοινώς max(a,b)
 - **printf(“To x %s μηδέν\n” , x==0 ? “είναι” : “δεν είναι”);**

Παράδειγμα - Άσκηση

- Να βρεθούν οι τιμές των x και y μετά την εκτέλεση καθεμιάς από τις παρακάτω εντολές, ξεκινώντας **πάντα** με τις αρχικές τιμές
- `int x=10;`
 - `int y=20;`

Εντολή	x	y
<code>++x;</code>	11	20
<code>y = x--;</code>	9	10
<code>y = --x;</code>	9	9
<code>y *= x--;</code>	9	200
<code>y += ++x;</code>	11	31
<code>y %= --x;</code>	9	2
<code>x *= 5</code>	50	20
<code>y = x *= 5</code>	50	50

Εκφράσεις

Εκφράσεις

□ Έκφραση

- Συνδυασμός ενός ή περισσότερων τελεστών και ενός ή περισσότερων τελεστών
- π.χ., $x = a + b$

□ Σημειογραφία Εκφράσεων

- **Infix notation** (ενδοθεματική): $x + y$
- **Prefix notation** (προθεματική): $+ x y$
- **Postfix notation** (μεταθεματική): $x y +$

Κατηγορίες Εκφράσεων

□ Αριθμητικές

- **Τελεστές:** αριθμητικοί
- **Τελεστέοι:** αριθμητικές τιμές, μεταβλητές, άλλες εκφράσεις
- **Αποτέλεσμα:** αριθμητικό
- Παράδειγμα: $(5 * x + y / 4) * 8$

□ Συγκριτικές

- **Τελεστές:** σύγκρισης
- **Τελεστέοι:** αριθμητικές τιμές, μεταβλητές, άλλες εκφράσεις
- **Αποτέλεσμα:** λογικού τύπου, ουσιαστικά int με τιμή 0 ή 1
- Παραδείγματα: $x > 5$, $a != b$, $x == 3$, $x + y >= 4$, $x - y > x + z$

□ Λογικές

- **Τελεστές:** λογικοί
- **Τελεστέοι:** παραστάσεις με τιμές λογικές
- **Αποτέλεσμα:** λογικού τύπου (0 ή 1)
- Παραδείγματα: $(x < 5) \&\& (x >= 1)$, $(x == 0) \mid \mid (y == 0)$

Υπολογισμός Εκφράσεων [1/3]

Η **εφαρμοστική σειρά** (**applicative order**) υπολογισμού των εκφράσεων βασίζεται σε δύο κανόνες:

1. Προτεραιότητα

- Πρώτα οι τελεστές ταξινομούνται ανά επίπεδο προτεραιότητας. Π.χ.:
 - στο $5+3*8$, το $*$ έχει προφανώς υψηλότερη προτεραιότητα από το $+$

2. Προσεταιριστικότητα

- Μετά, καθορίζεται η κατεύθυνση εφαρμογής μεταξύ των τελεστών ίδιας προτεραιότητας. Π.χ.:
 - στο $a+b+c$ θα υπολογιστεί πρώτα το $a+b$ και στη συνέχεια το $(a+b)+c$ (προσεταιριστικότητα από αριστερά προς δεξιά)
 - στο $a=b=c$ θα υπολογιστεί πρώτα το $b=c$ και στη συνέχεια το $a=(b=c)$ (προσεταιριστικότητα από δεξιά προς αριστερά)

Υπολογισμός Εκφράσεων [2/3]

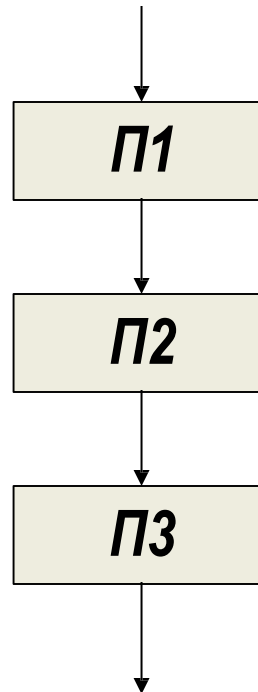
ΤΕΛΕΣΤΕΣ	ΠΡΟΣΕΤΑΙΡΙΣΤΙΚΟΤΗΤΑ
() [] -> ! ~ ++ -- - *_(pointer) & (τύπος) sizeof *_(πολλαπλασιασμός) / % + - << >> < <= > >= == != & ^ && ?: = += -= *= %= &= ^= = <<= >>= , (κόμμα)	Από αριστερά προς τα δεξιά Από δεξιά προς τα αριστερά Από αριστερά προς τα δεξιά Από αριστερά προς τα δεξιά Από αριστερά προς τα δεξιά Από αριστερά προς τα δεξιά Από αριστερά προς τα δεξιά Από αριστερά προς τα δεξιά Από αριστερά προς τα δεξιά Από αριστερά προς τα δεξιά Από αριστερά προς τα δεξιά Από δεξιά προς τα αριστερά Από δεξιά προς τα αριστερά Από αριστερά προς δεξιά

Υπολογισμός Εκφράσεων [3/3]

- Στη C ακολουθείται **υπολογισμός περιορισμένης έκτασης (short circuit evaluation)**
 - Δηλαδή όταν η τιμή μια έκφρασης είναι «προδικασμένη» δεν εκτελούνται όλοι οι υπολογισμοί μέχρι τέλους
 - Υπολογίζονται μόνο όσοι τελεστές απαιτούνται
 - Π.χ., στο **x && y**, αν το **x** βγει FALSE δεν θα γίνει ο υπολογισμός του **y**
 - Αντίστοιχα στο **x || y** αν το **x** βγει TRUE, μιας και το αποτέλεσμα θα είναι ούτως ή άλλως TRUE ανεξάρτητα από το **y**. Αν βέβαια το **x** βγει FALSE, θα υπολογιστεί και το **y** που θα κρίνει την τιμή της έκφρασης.
- Προσοχή: Πιθανές παρενέργειες!!
 - Το δεύτερο (ή τρίτο, κ.ο.κ.) σκέλος μιας λογικής έκφρασης μπορεί και να μην εκτελεστεί
 - Αν κάνει κάποια αλλαγή ή ανάθεση τιμής, *ίσως* να διαπιστώσουμε συμπεριφορά και αποτελέσματα που δεν περιμέναμε
 - π.χ. έστω ότι θέλουμε να μειώσουμε και το **x** και το **y** κατά 1, και να ελέγξουμε αν έχουν γίνει και τα δύο 0
 - Τότε, η έκφραση **(--x == 0) && (--y == 0)** περιλαμβάνει ένα καλά κρυμμένο ΛΑΘΟΣ. Γιατί;;;;;
 - Διότι το **y** θα μειωθεί μόνο αν το **x** μειούμενο γίνει 0, οπότε το αριστερό σκέλος υπολογίζεται σε TRUE, και έτσι η υπο-έκφραση μετά το && πρέπει να υπολογιστεί. Αλλιώς το δεξιό σκέλος θα παρακάμπτεται και το **y** θα παραμένει σταθερό.
- Γενικά, δεν εκτελείται η έκφραση **expr** στα παρακάτω σενάρια:
 - **FALSE && expr**
 - **TRUE || expr**

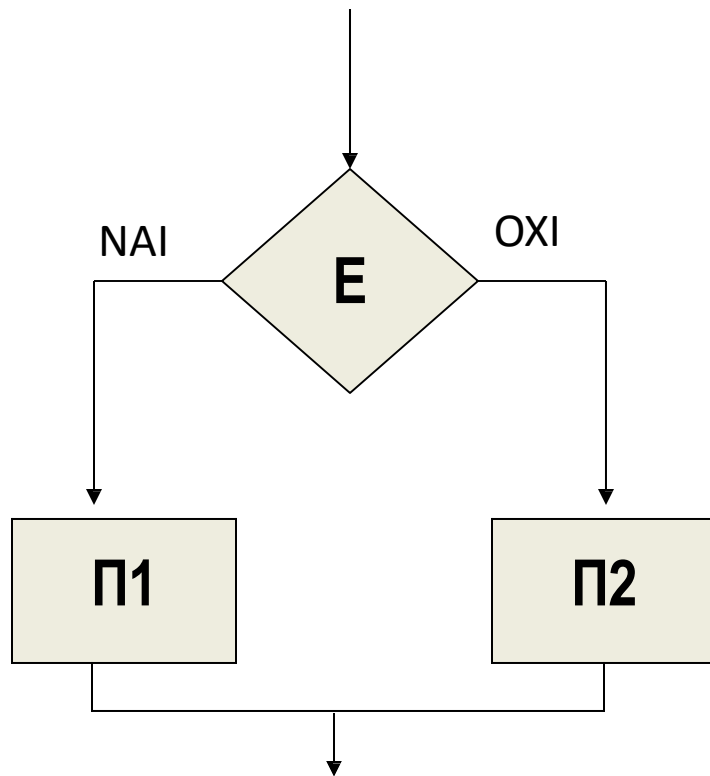
Έλεγχος Ροής

- Κανονικά, οι «προτάσεις» (εντολές) εκτελούνται σειριακά, κατ' ακολουθία



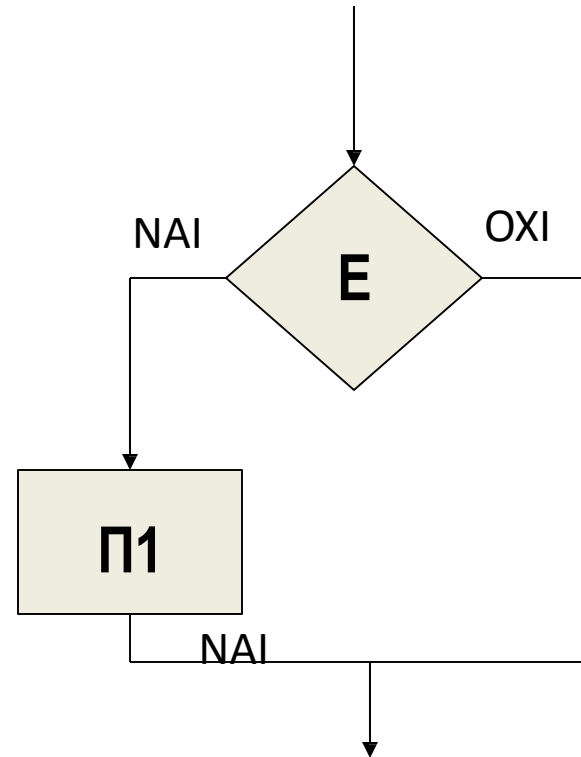
if: Πρόταση επιλογής

Απλή



```
if (E)
  Π1
else
  Π2
```

Περιορισμένη

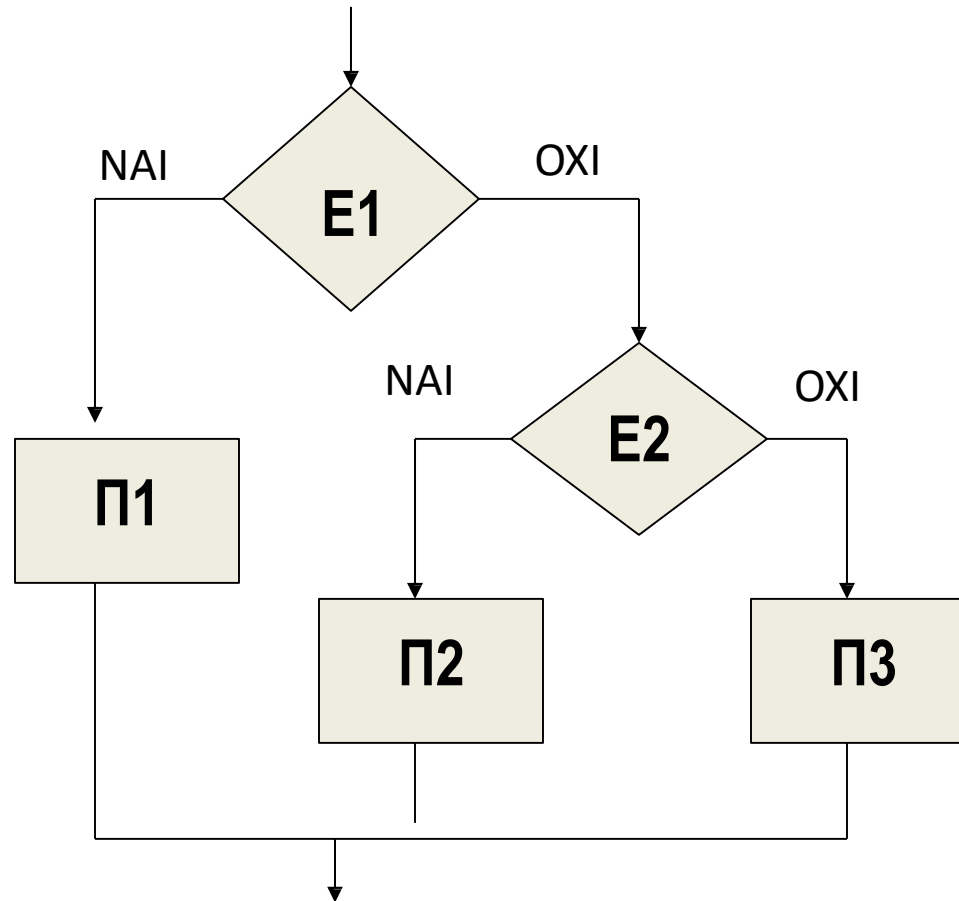


```
if
  (E)
  Π1
```

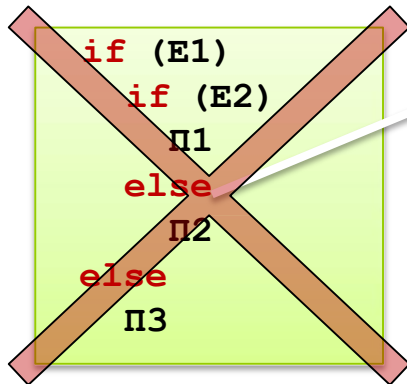
if: Πρόταση επιλογής

Εμφωλευμένη

```
if (E1)  
  Π1  
else if (E2)  
  Π2  
else  
  Π3
```



if: Πρόταση επιλογής



Σε ποιο if αντιστοιχεί αυτό το else;

ΠΡΟΣΟΧΗ!!!
Κάποιο πρόβλημα!!!

