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("Το 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
`++x;`	11	20
`y = x--;`	9	10
`y = --x;`	9	9
`y *= x--;`	9	200
`y += ++x;`	11	31
`y %= --x;`	9	2
`x *= 5`	50	20
`y = x *= 5`	50	50

Εκφράσεις

Εκφράσεις

- Έκφραση
  - Συνδυασμός ενός ή περισσότερων τελεστέων 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) || (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 κατεύθυνσης 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
() [ ] ->	Από αριστερά προς τα δεξιά
! ~ ++ -- - $*_{(pointer)}$ & (τύπος) sizeof	Από δεξιά προς τα αριστερά
$*_{(πολλαπλασιασμός)}$ / %	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 αριστερά προς τα δεξιά
?:	Από δεξιά προ τα αριστερά
= += -= *= %= &= ^= |= <<= >>=	Από δεξιά προς τα αριστερά
, (κόμμα)	Από αριστερά προς δεξιά

Από: Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C KERNIGHAN W. BRIAN, RITCHIE M. DENNIS, εκδόσεις Κλειδάριθμος, δεύτερη έκδοση

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**

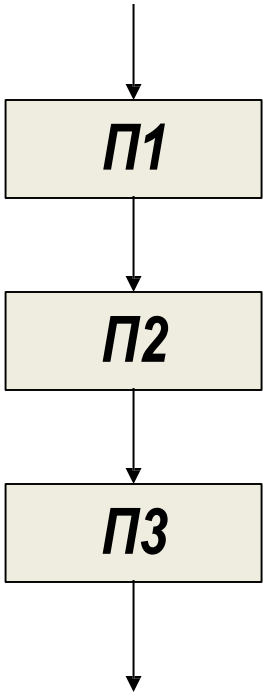
Προτάσεις ελέγχου ροής

Δομή διάλεξης

- Έλεγχος ροής
- Πρόταση επιλογής (if, switch)
- Προτάσεις επανάληψης (while, do-while, for)
- Διακλάδωση χωρίς συνθήκη (break, continue, goto)

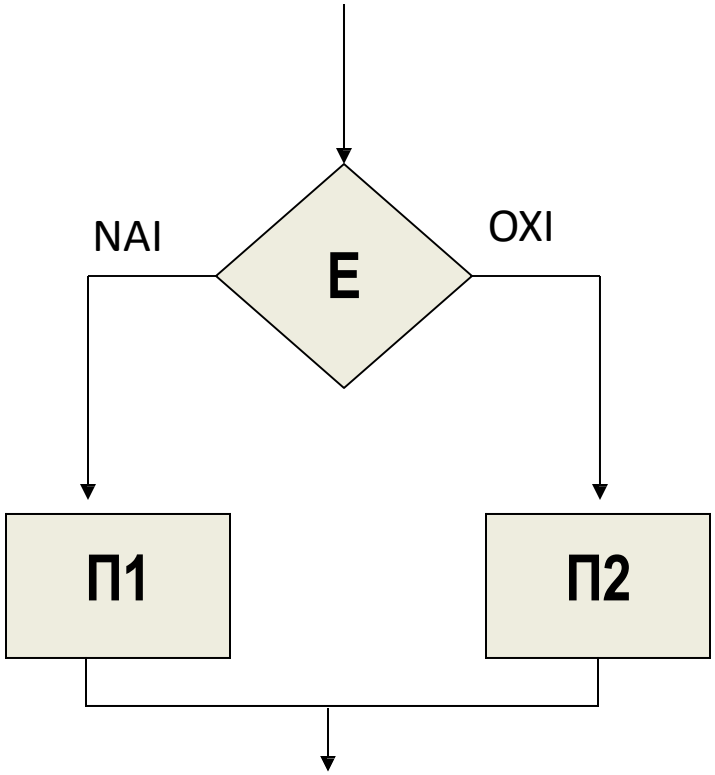
Έλεγχος Ροής

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



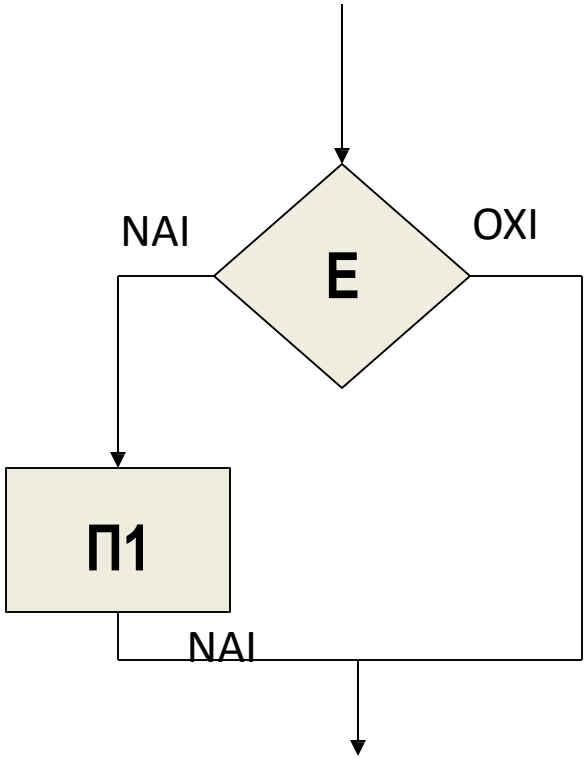
if: Πρόταση επιλογής

Απλή




```
if (E)
   Π1
else
   Π2
```

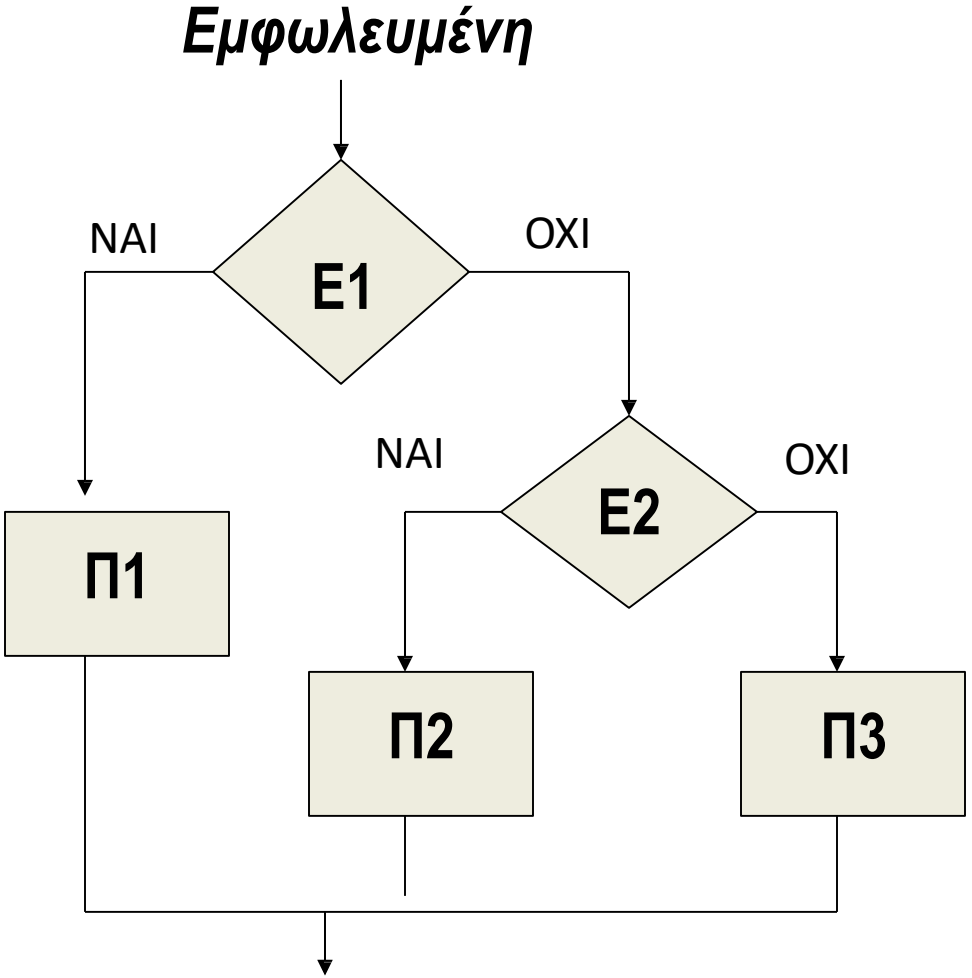
Περιορισμένη




```
if
   (E)
   Π1
```

if: Πρόταση επιλογής

```
if (Ε1)
  Π1
else if (Ε2)
  Π2
else
  Π3
```



if: Πρόταση επιλογής

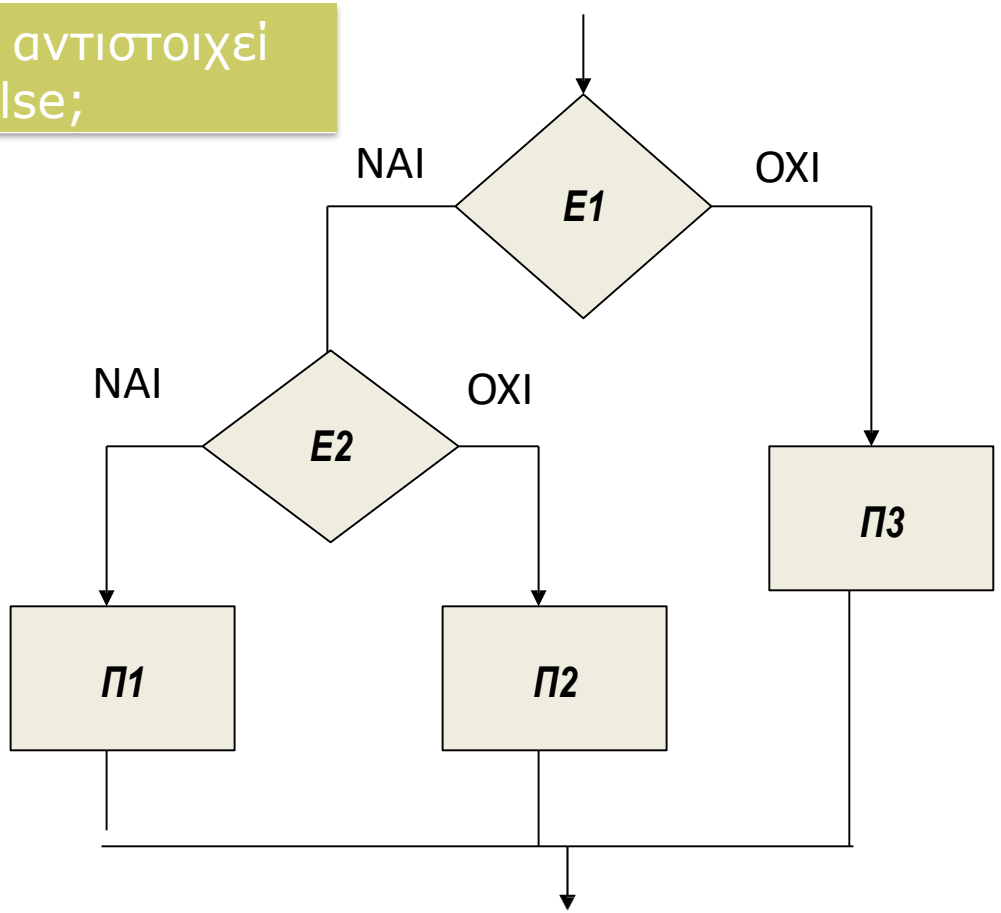
```
if (E1)
  if (E2)
    Π1
  else
    Π2
else
  Π3
```

Σε ποιο if αντιστοιχεί αυτό το else;

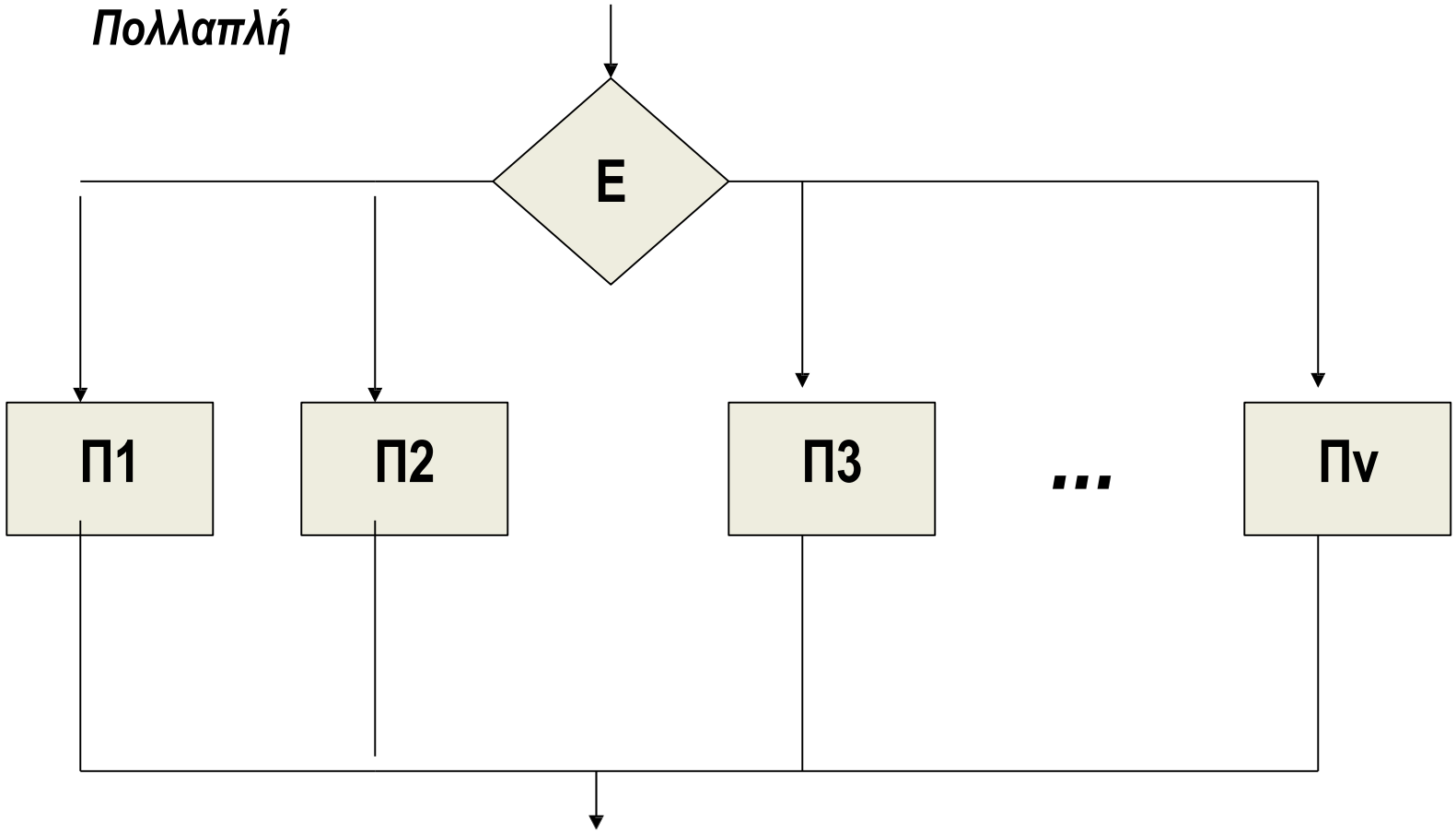
ΠΡΟΣΟΧΗ!!!
Κάποιο πρόβλημα;;;

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

Εμφωλευμένο



if: Πρόταση επιλογής



if: Πρόταση επιλογής

```
if  (<έκφραση>)
    <πρόταση1>
[else  <πρόταση2>]
```

απλή ή σύνθετη

```
If  (<έκφραση>)
    <πρόταση1>
  else  if  (<έκφραση2>)
          <πρόταση2>
      else if (<έκφραση3>)
            <πρόταση3>
          else  <πρόταση4>
```

Εμφωλευμένο if

Παράδειγμα

```
#include <stdio.h>
main()
{
  float num;
  printf("Δώσε αριθμό: ");
  scanf("%f", &num);
  if (num < 0)
    printf("Η απόλυτη τιμή του %f είναι: %f\n", num, -num);
  else
    printf("Η απόλυτη τιμή του %f είναι: %f\n", num, num);

  printf("Η απόλυτη τιμή: %f\n", num < 0 ? -num : num);
}
```

switch: Πολλαπλή επιλογή

- Όταν βάσει μιας έκφρασης θέλουμε να επιλέξουμε ανάμεσα σε πολλές επιλογές, είναι πιο «βολική» η χρήση της **switch**.

```
switch (<έκφραση>)
{
  case <σταθ-εκφρ-1>: <προτ-1>; break;
  case <σταθ-εκφρ-2>: <προτ-2>; break;
      …
  case <σταθ-εκφρ-N>: <προτ-N>; break;
  default: <πρόταση>; break;
}
```

switch: Πολλαπλή επιλογή

- Κάθε <σταθ-έκφρ-i> πρέπει να είναι **μία τιμή int ή char** ή μία έκφραση μόνο με τέτοιες τιμές
- Δύο <σταθ-εκφρ-i> δεν μπορούν να έχουν την ίδια τιμή
- Αν <έκφραση>=<σταθ-εκφρ-x> τότε εκτελούνται όλες οι παρακάτω της x προτάσεις
  - ✓ Για να το αποτρέψουμε αυτό, χρειάζεται η **break**!
- Η <πρόταση> εκτελείται μόνο όταν καμμιά από τις <προτ-i> δεν ικανοποιείται
- Δεν υποστηρίζονται περιοχές τιμών (ranges), μόνο ισότητα
- Η default δεν είναι απαραίτητο να είναι στο τέλος

Παράδειγμα

```
switch (choice)   {
      case 1:
        x=a+b;
        break;
      case 2:
        x=a-b;
        break;
      case 3:
        x=a*b;
        break;
      case 4:
        x=a/b;
        break;
      default:
        printf("Ανύπαρκτη επιλογή");
        break;
}
```

Παράδειγμα

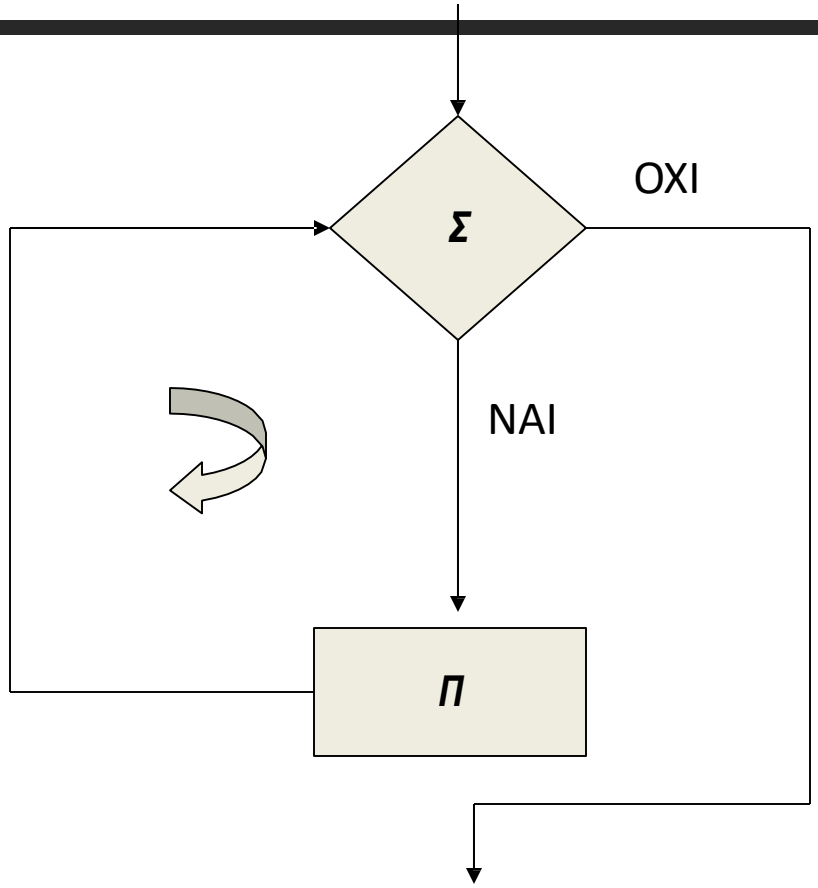
```
switch (choice) {
      case 1: x=a+b; break;
      case 2: x=a-b; break;
      case 3: x=a*b;break;
      case  4:  x=a/b;break;
      default:
        printf("Ανύπαρκτη επιλογή");
        break;
}
```

Loops (Βρόχοι)

- Επανάληψη τμήματος κώδικα
  - ✓ για έναν αριθμό επαναλήψεων
  - ✓ ή γενικά μέχρι να ισχύσει μία συνθήκη

while: Loop με Συνθήκη Εισόδου

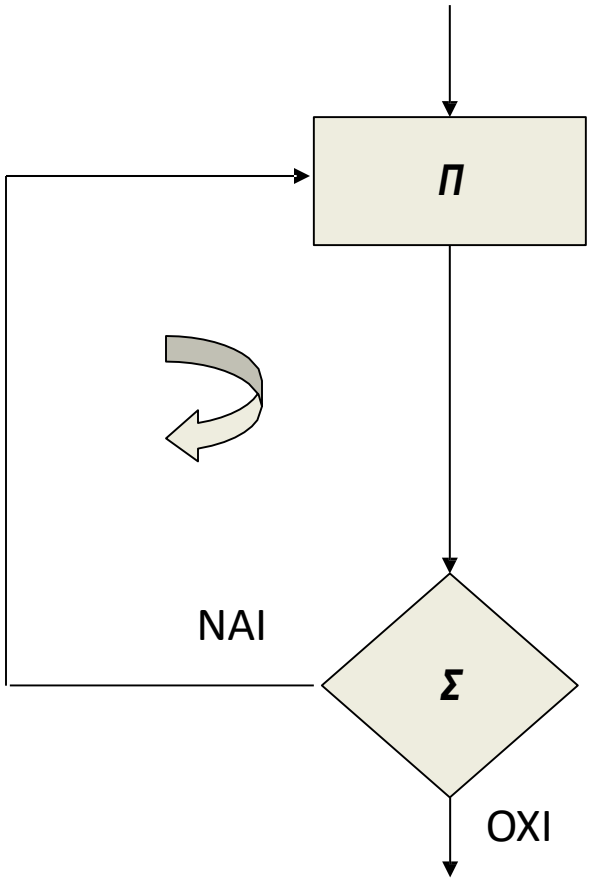
```
while (Σ)
  Π
```



do-while: Loop με Συνθήκη Εξόδου

```
do
  Π
while (Σ)
```

Το Π θα εκτελεστεί τουλάχιστον μία φορά



Loops με μετρητή

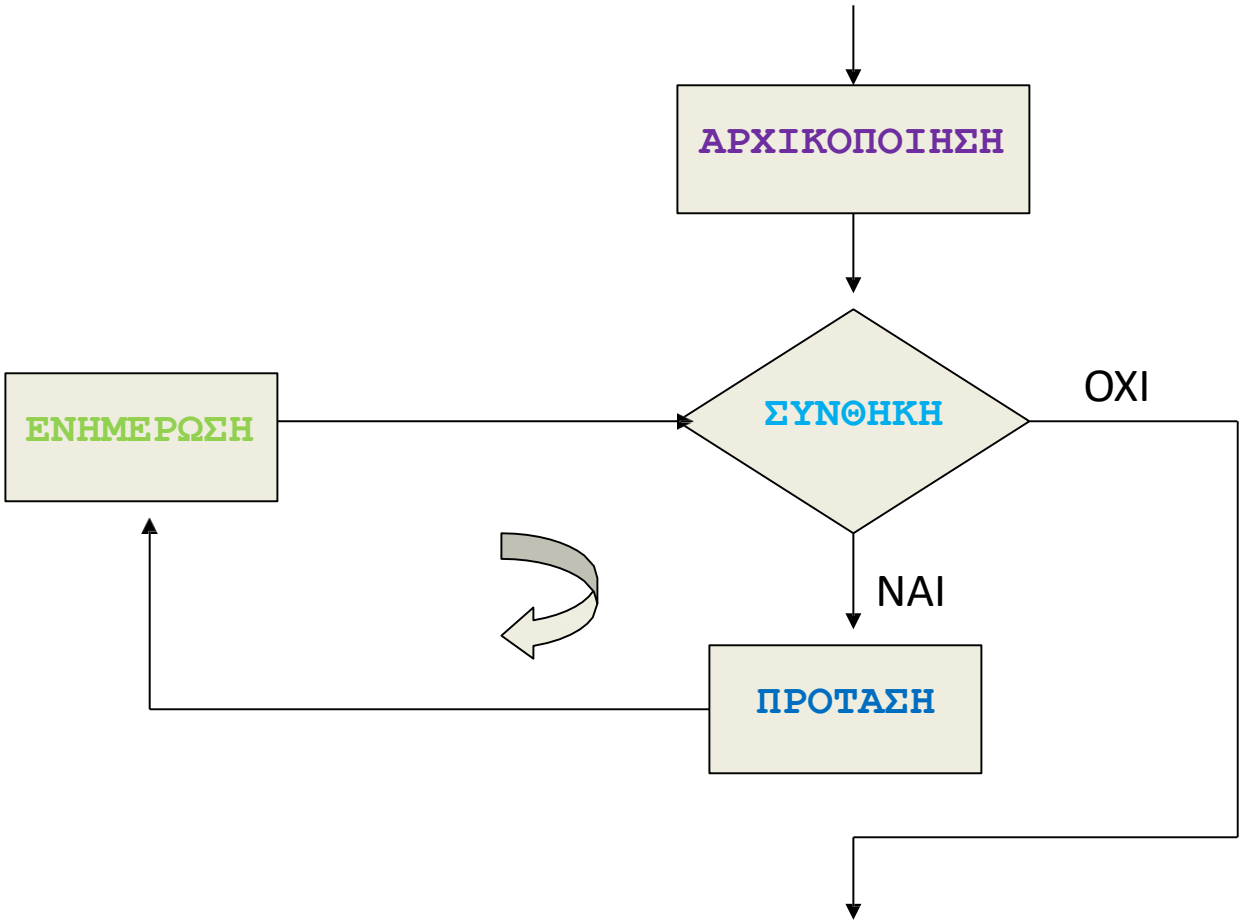
- Πολλές φορές θέλουμε ένα loop να εκτελεστεί για έναν συγκεκριμένο αριθμό επαναλήψεων
- Χρησιμοποιούμε έναν μετρητή

```
// Για να μετρήσουμε απ΄το 0 ως το 9

int counter = 0;                          ΑΡΧΙΚΟΠΟΙΗΣΗ

while (counter < 10)                      ΣΥΝΘΗΚΗ
{
  printf("Ο counter είναι %d\n", counter);
  counter++;                              ΕΝΗΜΕΡΩΣΗ
}
```

- Πιο εύκολα με την εντολή **for**

for: Loop με μετρητή

```
for (ΑΡΧΙΚΟΠΟΙΗΣΗ; ΣΥΝΘΗΚΗ; ΕΝΗΜΕΡΩΣΗ)
  ΠΡΟΤΑΣΗ
```



for: Loop με μετρητή

- Απαρίθμηση από 0 ως n-1: `for (i=0; i<n; i++)`
- Απαρίθμηση από 1 ως n: `for (i=1; i<=n; i++)`
- Απαρίθμηση από n-1 ως 0: `for (i=n-1; i>=0; i--)`
- Απαρίθμηση από n ως 1: `for (i=n; i>=1; i--)`
- Δυνάμεις του 2 μέχρι 1e6: `for (i=1; i<1e6; i*=2)`

Επιλογή Πρότασης Επανάληψης

- Προτιμούμε την πρόταση με συνθήκη εισόδου (while) από αυτή με συνθήκη εξόδου (do-while)

- Προτιμούμε τη for από τη while, αν υπάρχει (ή μπορεί να οριστεί) απαριθμητής που συνοδεύεται από αρχικοποίηση και ανανέωση της τιμής του

Διακλάδωση χωρίς συνθήκη

- Διαχείριση ειδικών περιπτώσεων σε προτάσεις επανάληψης
  - ✓ break
  - ✓ continue
- Ρητή διακλάδωση
  - ✓ goto <ετικέτα>

break: Διακοπή loop

- Την είδαμε στη switch
- Γενικά, προκαλεί την έξοδο μόνο από τον πιο εσωτερικό βρόχο
- Προσοχή!
  - ✓ Καταστρέφει τη δόμηση του κώδικα
  - ✓ Υπάρχει πάντα τρόπος να γραφεί κώδικας χωρίς τη χρήση της

```
while (<έκφραση>)
{
  if (ειδική περίπτωση)
  {
    προτάσεις επεξεργασίας ειδικής περίπτωσης;
    break;
  }
  προτάσεις επεξεργασίας κανονικών περιπτώσεων;
}
```

continue: Παράκαμψη επανάληψης

- Παρακάμπτει την τρέχουσα επανάληψη, περνώντας άμεσα στην επόμενη
- Επηρεάζει μόνο τον πιο εσωτερικό βρόχο

- Προσοχή!
  - ✓ Καταστρέφει τη δόμηση του κώδικα
  - ✓ Υπάρχει πάντα τρόπος να γραφεί κώδικας χωρίς τη χρήση της

```
while (<έκφραση>)
{
  if (κανονική περίπτωση)
  {
    προτάσεις επεξεργασίας κανονικής περίπτωσης;
    continue;
  }
  προτάσεις επεξεργασίας ειδικών περιπτώσεων;
}
```

goto: Αυθαίρετη διακλάδωση

- Μεταφέρει τη ροή σε αυθαίρετο σημείο του κώδικα

- Προσοχή!
  - ✓ Πολύ επικίνδυνη
  - ✓ Πολύ κακή τεχνική
  - ✓ «Απαγορεύεται» η χρήση της!!☺

```
...
start:
...
...
...
goto start;
...
```