

Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Εαρινό Εξάμηνο 2022

Ασκήσεις

Σύνολο 2Α

Εμβέλεια – Διαχείριση Μνήμης – Υποπρογράμματα

Γ. Γαροφαλάκης

ΑΣΚΗΣΗ 1

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal στην οποία ισχύει ο στατικός κανόνας εμβέλειας:

```
program MAIN;  
  var i, a: integer;  
  function F(j, b, c, d: integer): integer;  
    var e: integer;  
    begin  
      e:= 1;  
      for j:= b to c do e:= e*d;  
      F:= e  
    end;  
  BEGIN  
    i:= 1;  
    e:= i;  
    a:= F(i, 1, 10, i);  
    write(a)  
  END.
```

- Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;
- Στον παραπάνω κώδικα υπάρχει μία εντολή η οποία δεν είναι σωστή από σημασιολογική άποψη και θα προκαλέσει run-time error. Ποια είναι αυτή και γιατί; «Σβήστε» την εντολή αυτή από τον κώδικα.
- Τι υπολογίζει η συνάρτηση F και τυπώνεται στο τέλος με την εντολή `write(a)`, στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων (παρουσιάστε και εξηγήστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού);
 - Κλήση με τιμή (call by value)
 - Κλήση με αναφορά (call by reference)

Δώστε το αποτέλεσμα σας είτε με αριθμό, είτε με μαθηματικό τύπο.

ΑΠΑΝΤΗΣΕΙΣ ΑΣΚΗΣΗΣ 1

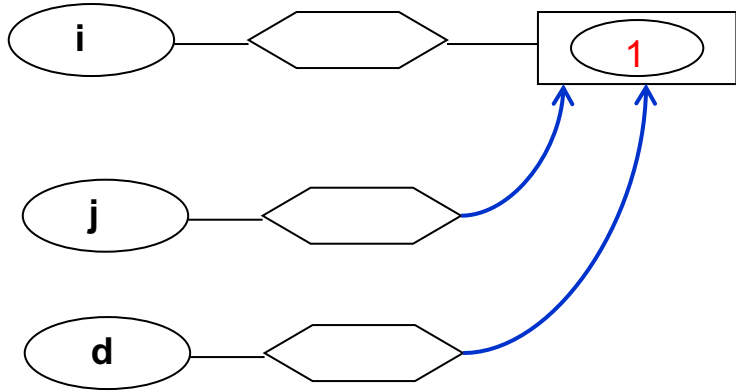
- a) **MAIN**: - Τοπικό ΠΑ: i , a , F (κλήση)
- Μη-τοπικό και Καθολικό ΠΑ: i , a , F (κλήση)
- F**: - Τοπικό ΠΑ: j , b , c , d , e , F (τιμή)
- Μη-τοπικό και Καθολικό ΠΑ: i , a , F (κλήση)

```
program MAIN;  
  var i, a: integer;  
  function F(j, b, c, d: integer): integer;  
    var e: integer;  
    begin  
      e:= 1;  
      for j:= b to c do e:= e*d;  
      F:= e  
    end;  
BEGIN  
  i:= 1;  
  e:= i;  
  a:= F(i, 1, 10, i);  
  write(a)  
END.
```

- b) Η εντολή **$e := i$** ; δεν είναι σημασιολογικά σωστή: Το **e** δεν είναι ορατό στο MAIN. Όπως φαίνεται παραπάνω, δεν είναι στα Περιβάλλοντα Αναφοράς του MAIN.
- c) i) **Call by value**
Οι τυπικές παράμετροι j , b , c , d παίρνουν τις τιμές των πραγματικών παραμέτρων $i=1$, 1 , 10 , $i=1$ αντίστοιχα στην αρχή εκτέλεσης της F και δεν συνδέονται πλέον. Δηλαδή, εκτελείται στην F η εντολή: **for $j := 1$ to 10 do $e := e * 1$** ;
Έχουμε $e := 1$; Οπότε $F = e = 1$ και τελικά **write (a) \rightarrow 1**

ii) Call by reference

Τα j , d είναι τώρα pointers στο i .



Συνεπώς, όταν το j αυξάνεται κατά 1, αυξάνεται κατά 1 το i , οπότε αυξάνεται κατά 1 και το d .

Δηλαδή, στην F εκτελείται ουσιαστικά η εντολή:

```
for j:= 1 to 10 do e:= e*j;
```

Άρα, το τελικό αποτέλεσμα είναι

$$\text{write (a)} \rightarrow \prod_{i=1}^{10} i = 3.628.800$$

```
program MAIN;  
  var i, a: integer;  
  function F(j, b, c, d: integer): integer;  
    var e: integer;  
    begin  
      e:= 1;  
      for j:= b to c do e:= e*d;  
      F:= e  
    end;  
BEGIN  
  i:= 1;  
  e:= i;  
  a:= F(i, 1, 10, i);  
  write(a)  
END.
```

ΑΣΚΗΣΗ 2

Δίνεται το παρακάτω πρόγραμμα σε μια γλώσσα τύπου Pascal στην οποία ισχύει ο στατικός κανόνας εμβέλειας:

```
program MAIN;  
    var z: integer;  
        a: array [1..2] of integer;  
procedure P(x: integer);  
    begin  
        a[1]:= 6;  
        z:= 2;  
        x:= x + 5  
    end;  
BEGIN  
    a[1]:= 2;  a[2]:=3;  
    z:= 1;  x:= 2;  
    P(a[z]);  
    write(a[1], a[2], z)  
END.
```

- a) Ποια είναι τα περιβάλλοντα αναφοράς (τοπικά, μη-τοπικά, καθολικά) όλων των τμημάτων του προγράμματος;
- b) Στον παραπάνω κώδικα υπάρχει/ουν εντολή/ές που δεν είναι σωστή/ές από σημασιολογική άποψη και θα προκαλέσει/ουν run-time error. Ποια/ες είναι αυτή/ες και γιατί; «Σβήστε» την/τις εντολή/ες από τον κώδικα.
- c) Τι τυπώνεται στο τέλος με την εντολή **write**, στις παρακάτω περιπτώσεις τρόπου μεταβίβασης παραμέτρων (παρουσιάστε και εξηγήστε τις αλλαγές τιμών όλων των μεταβλητών κατά τη διαδικασία υπολογισμού);
 - i. Κλήση με τιμή (call by value)
 - ii. Κλήση με τιμή – αποτέλεσμα (call by value – result)
 - iii. Κλήση με αναφορά (call by reference)
 - iv. Κλήση με όνομα (call by name)

ΑΠΑΝΤΗΣΕΙΣ ΑΣΚΗΣΗΣ 2

- a) **MAIN:** - Τοπικό ΠΑ: z, a, P (κλήση)
- Μη-τοπικό και Καθολικό ΠΑ: z, a, P (κλήση)

P: - Τοπικό ΠΑ: x

- Μη-τοπικό και Καθολικό ΠΑ: z, a, P (κλήση)

- b) Η μόνη εντολή που δεν είναι σημασιολογικά σωστή, είναι η **x := 2;** : Το **x** δεν είναι ορατό στο MAIN. Όπως φαίνεται παραπάνω, δεν είναι στα Περιβάλλοντα Αναφοράς του MAIN.

```
program MAIN;
    var z: integer;
        a: array [1..2] of integer;
    procedure P(x: integer);
        begin
            a[1]:= 6;
            z:= 2;
            x:= x + 5;
        end;
    BEGIN
        a[1]:= 2; a[2]:=3;
        z:= 1; x:= 2;
        P(a[z]);
        write(a[1], a[2], z)
    END.
```

c) i) Call by value

- Κλήση της *P*: $P(a[z]) = P(a[1]) = P(2) \longrightarrow x=2$ (ανεξάρτητο του $a[1]$)

- Εκτέλεση της *P*: $a[1] := 6; z := 2; x := x + 5 = 2 + 5 = 7$

- Επιστροφή στο *MAIN*: `write (a[1], a[2], z) → 6, 3, 2`

ii) Call by value – result

- Κλήση της *P*: $P(a[z]) = P(a[1]) = P(2)$

$\longrightarrow x=2$ (ανεξάρτητο του $a[1]$)

- Εκτέλεση της *P*:

$a[1] := 6; z := 2; x := x + 5 = 2 + 5 = 7$

- Επιστροφή στο *MAIN*:

Η τρέχουσα τιμή του x αντιγράφεται στην τιμή του $a[1]$. Δηλαδή $x = 7 \longrightarrow a[1] = 7$

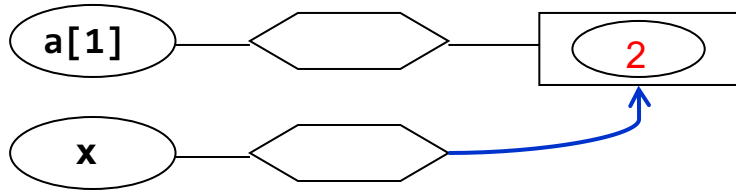
και `write (a[1], a[2], z) → 7, 3, 2`

```
program MAIN;
    var z: integer;
        a: array [1..2] of integer;
    procedure P(x: integer);
        begin
            a[1] := 6;
            z := 2;
            x := x + 5;
        end;
BEGIN
    a[1] := 2; a[2] := 3;
    z := 1; x := 2;
    P(a[z]);
    write(a[1], a[2], z)
END.
```

iii) Call by reference

- Κλήση της P : $P(a[z]) = P(a[1])$

Το x είναι τώρα pointer στο $a[1]$



$a[1]=2 \iff x=2$

- Εκτέλεση της P : $a[1] := 6 \implies x = 6$;

$z := 2$;

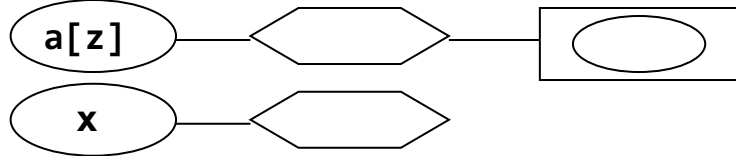
$x := x + 5 = 6 + 5 = 11 \implies a[1] := 11$

- Επιστροφή στο $MAIN$: **write (a[1], a[2], z) \rightarrow 11, 3, 2**

```
program MAIN;
    var z: integer;
        a: array [1..2] of integer;
    procedure P(x: integer);
        begin
            a[1]:= 6;
            z:= 2;
            x:= x + 5;
        end;
    BEGIN
        a[1]:= 2; a[2]:=3;
        z:= 1; x:=2;
        P(a[z]);
        write(a[1], a[2], z)
    END.
```


iv) Call by name

- Κλήση της P : $P(a[z])$



Το $a[z]$ δεν υπολογίζεται στην κλήση, αλλά κάθε φορά που χρησιμοποιείται το x στην P . Τότε συνδέεται το x με το $a[z]$

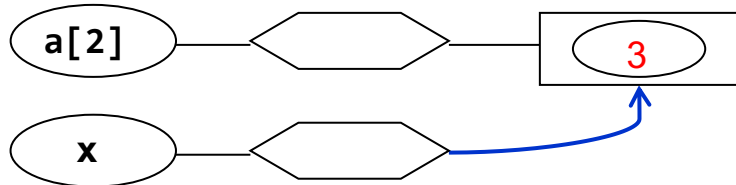
- Εκτέλεση της P :

$a[1] := 6;$

$z := 2;$

$x := x + 5$ (?)

Τώρα θα υπολογιστεί το $a[z] = a[2] = 3$ και θα γίνει η σύνδεση του x με το $a[2]$



$x := x + 5 = a[2] + 5 = 3 + 5 = 8 \longrightarrow a[2] := 8$

- Επιστροφή στο $MAIN$: $a[1] := 6;$ $a[2] := x = 8$

και

$write(a[1], a[2], z) \rightarrow 6, 8, 2$

```
program MAIN;
    var z: integer;
        a: array [1..2] of integer;
    procedure P(x: integer);
        begin
            a[1] := 6;
            z := 2;
            x := x + 5;
        end;
    BEGIN
        a[1] := 2; a[2] := 3;
        z := 1; x := 2;
        P(a[z]);
        write(a[1], a[2], z)
    END.
```

ΑΣΚΗΣΗ 3

Δίνεται το παρακάτω τμήμα προγράμματος C++:

1	<code>int main()</code>
2	<code>{</code>
3	<code>int numbers[5];</code>
4	<code>int *p;</code>
5	<code>p = numbers; *p = 10;</code>
6	<code>p = numbers + 3; *p = 40;</code>
7	<code>p = &numbers[2]; *p = 30;</code>
8	<code>p = numbers; *(p+4) = 50;</code>
9	<code>p++; *p = 20;</code>
10	<code>for (int n=0; n<5; n++)</code>
11	<code>cout << numbers[n] << ", ";</code>
12	<code>return 0;</code>
13	<code>}</code>

- Εξηγήστε γραμμή – γραμμή τους ορισμούς και τη λειτουργία του παραπάνω προγράμματος.
- Τι θα τυπωθεί με την εκτέλεση του προγράμματος;

ΑΠΑΝΤΗΣΕΙΣ ΑΣΚΗΣΗΣ 3

a)

1	<code>int main()</code>	
2	<code>{</code>	
3	<code>int numbers[5];</code>	Ορίζεται ένα array (numbers) 5 ακεραίων από τη stack memory
4	<code>int *p;</code>	Ορίζεται ένας pointer (p)
5	<code>p = numbers; *p = 10;</code>	Το p δείχνει στο numbers[0] . Το numbers[0] παίρνει την τιμή 10 .
6	<code>p = numbers + 3; *p = 40;</code>	Το p δείχνει τώρα στο numbers[3] . Το numbers[3] παίρνει την τιμή 40 .
7	<code>p = &numbers[2]; *p = 30;</code>	Το p δείχνει στο numbers[2] . Το numbers[2] παίρνει την τιμή 30 .
8	<code>p = numbers; *(p+4) = 50;</code>	Το p δείχνει πάλι στο numbers[0] . Το numbers[4] παίρνει την τιμή 50 .
9	<code>p++; *p = 20;</code>	Το p δείχνει στο numbers[1] . Το numbers[1] παίρνει την τιμή 20 .
10	<code>for (int n=0; n<5; n++)</code>	Τυπώνονται οι τιμές numbers[0] – numbers[4]
11	<code>cout << numbers[n] << ", ";</code>	
12	<code>return 0;</code>	
13	<code>}</code>	

b) **10, 20, 30, 40, 50**

ΑΣΚΗΣΗ 4

a)

- i. Δίνεται η εντολή της **C**: **if A < B || C < D then ...** Δείξτε τη σειρά εκτέλεσης των υπολογισμών χρησιμοποιώντας παρενθέσεις.
- ii. Κάντε το ίδιο για την αντίστοιχη εντολή στην **Pascal**: **if A < B or C < D then ...**
- iii. Για τις δύο παραπάνω περιπτώσεις, υπάρχουν κάποιες προϋποθέσεις για τους τύπους των μεταβλητών **A, B, C, D**, ώστε να μην επιστραφεί λάθος κατά τη μετάφραση;

- b) Στην παρακάτω έκφραση της **C**, χρησιμοποιούνται ακέραιες μεταβλητές με τρέχουσες τιμές:
 $A = 1, B = 2, C = 3, D = 4, E = 5$:

A || B < C && D + (E = 1)

Ποια τιμή έχει η έκφραση; Αριθμήστε τη χρονική ακολουθία των πράξεων στην έκφραση, χρησιμοποιώντας και παρενθέσεις.

ΑΠΑΝΤΗΣΕΙΣ ΑΣΚΗΣΗΣ 4

a)

i. **C:** `if (A < B) || (C < D) then ...`

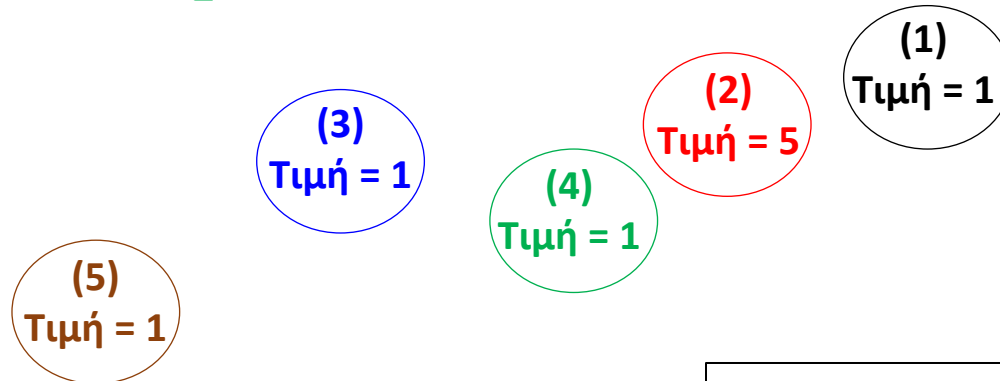
ii. **Pascal:** `if (A < (B or C)) < D then ...`

iii. - Στη **C** δεν υπάρχουν περιορισμοί. Όλοι οι υπολογισμοί και συγκρίσεις, είναι ουσιαστικά μεταξύ integers.

- Στην **Pascal** θα πρέπει όλα (A, B, C, D) να είναι τύπου Boolean. Αλλιώς, λάθος.

b)

`{ A || [(B < C) && (D + (E = 1))] }`



Δηλαδή, η έκφραση έχει την τιμή **1**.

`A || B < C && D + (E = 1)`