

Κεφάλαιο 7: Υποπρογράμματα

Αρχές Γλωσσών Προγραμματισμού και Μεταφραστών

Γ. Γαροφαλάκης, Σ. Σιούτας, Π. Χατζηδούκας

Ορισμός

Αφαίρεση με χρήση υποπρογραμμάτων

(subprogram abstraction) είναι η αντιστοίχιση ενός συνόλου εισόδων σε ένα σύνολο εξόδων που μπορεί να περιγραφεί φορμαλιστικά.

Η περιγραφή της χρήσης πρέπει να δείχνει πως σχετίζονται οι έξοδοι με τις εισόδους, αλλά δεν χρειάζεται να δείχνει τον τρόπο με τον οποίο υπολογίζονται οι έξοδοι.

Ο προγραμματιστής εστιάζει την προσοχή του στο τι γίνεται στο σημείο της κλήσης, και όχι στον τρόπο με τον οποίο γίνεται.

Χαρακτηριστικά Υποπρογραμμάτων

- Κάθε υποπρόγραμμα (ΥΠ) έχει ένα μόνο σημείο εισόδου.
- Η καλούσα μονάδα προγράμματος αναστέλλεται κατά την εκτέλεση του ΥΠ, οπότε υπάρχει μόνο ένα ΥΠ υπό εκτέλεση σε κάθε χρονική στιγμή.
- Ο έλεγχος επιστρέφει πάντα στην καλούσα μονάδα όταν ολοκληρώνεται η εκτέλεση του ΥΠ.
- Τις περισσότερες φορές, τα ΥΠ έχουν όνομα.

Είδη Υποπρογραμμάτων

- **Διαδικασία** (procedure)

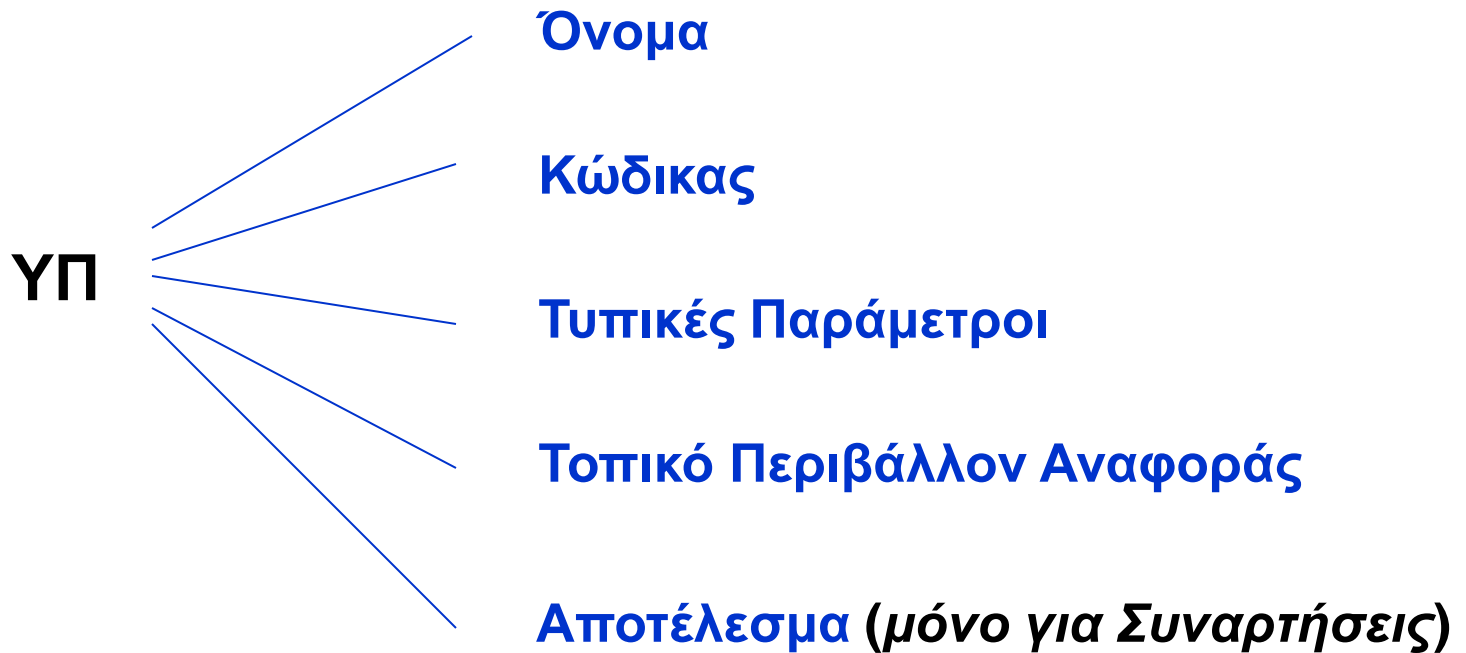
Εκπληρώνει το έργο της, είτε εκχωρώντας τα αποτελέσματά της σε μία ή περισσότερες από τις παραμέτρους της, είτε αλλάζοντας το περιβάλλον της (τιμές μη-τοπικών μεταβλητών, ΠΑ), είτε κάνοντας και τα δύο.

- **Συνάρτηση** (function)

Είναι Διαδικασία που, επιπλέον, επιστρέφει μία τιμή.

Συστατικά Υποπρογραμμάτων

Ένα Υποπρόγραμμα, περιλαμβάνει
4 (διαδικασίες), ή 5 (συναρτήσεις) **στοιχεία** :



Και άλλοι Ορισμοί

Τυπικές Παράμετροι: Δεν είναι ακριβώς μεταβλητές, αλλά ονόματα, που μπορεί να γίνουν τοπικές μεταβλητές και δείχνουν το ρόλο που θα παίξουν οι πραγματικές παράμετροι, όταν κληθεί η υπορουτίνα.

Πραγματικές Παράμετροι: Οι μεταβλητές και/ή οι εκφράσεις που παρέχονται σε μια υπορουτίνα, για να αντικαταστήσουν τις τυπικές παραμέτρους.

Παραδείγματα (1)

■ Pascal:

```
PROCEDURE F(X: real; var Y: integer) [ : real ]
```

```
[FUNCTION]
```

```
    VAR M: array[1..10] of real;
```

```
        N: integer
```

```
begin
```

```
    ...
```

```
end;
```

Παραδείγματα (2)

- C functions:

```
float power(float base, float exp)
```

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

Κλήση:

```
x = power(10.0, x)
```

- C procedures:

```
void sort(int x[ ], int a)
```

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

Κλήση:

```
sort(scores, 100)
```


Σχεδιαστικά Θέματα (1)

- **Λίστα Παραμέτρων ή Προσδιορισμός Παραμέτρων**

Εκτός από το **όνομα** των παραμέτρων, περιλαμβάνει τον **τύπο** και τον **τρόπο** χρήσης. Π.χ.

PROCEDURE F(X: real; var Y: integer)

- **Αντιστοιχία Τυπικών – Πραγματικών Παραμέτρων**

Με βάση τη σειρά αναγραφής στον ορισμό της υπορουτίνας.

Εξαίρεση: Η **Ada** επιτρέπει την ανατροπή της σειράς. Π.χ.

Ορισμός: **F(A, B)**. Κλήση: **F(B->100, A->10)**.

Σχεδιαστικά Θέματα (2)

- Είδος τιμών που επιστρέφει μια Συνάρτηση

FORTRAN, ALGOL60: Απλοί βαθμωτοί ΤΔ (real, int, bool)

PL/1: Βαθμωτοί και αλφαριθμητικά και pointers

Pascal: βαθμωτοί και pointers

C, Ada: Όλους τους ΤΔ.

Στις περισσότερες ΓΠ, πρέπει να εκχωρηθεί τιμή στο όνομα της function, πριν το τέλος της περιγραφής της. Π.χ. **Pascal:**

```
function f(x: integer): integer
```

```
begin
```

```
  if x<=1 then
```

```
    f:=1
```

```
  else
```

```
    f:=x*f(x-1)
```

```
end;
```

Υπολογισμός και Μεταβίβαση Παραμέτρων

■ Υπολογισμός Παραμέτρων (parameter evaluation)

Διεργασία κατά την οποία κάθε πραγματική παράμετρος αναγνωρίζεται ότι συνδέεται με την αντίστοιχη τυπική παράμετρο, και μετά υπολογίζεται.

■ Μεταβίβαση Παραμέτρων (parameter passing)

Ο τρόπος με τον οποίο η υπολογισμένη πραγματική παράμετρος μεταφέρεται (συνδέεται) στο ΥΠ.

- Κλήση με Τιμή (call by value)
- Κλήση με Αναφορά (call by reference)
- Κλήση με Τιμή - Αποτέλεσμα (call by value – result)
- Κλήση με Όνομα (call by name)
-

Κλήση με Τιμή (1)

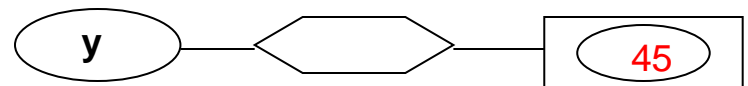
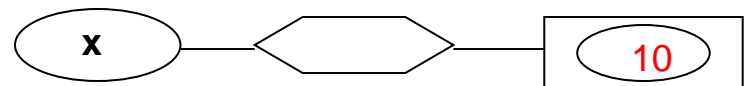
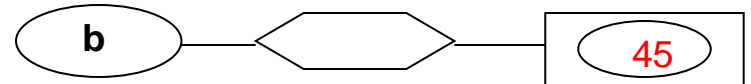
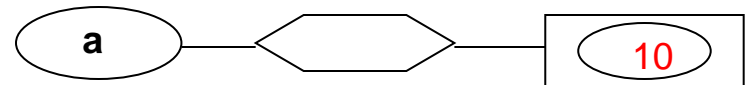
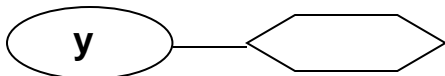
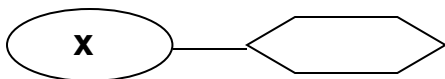
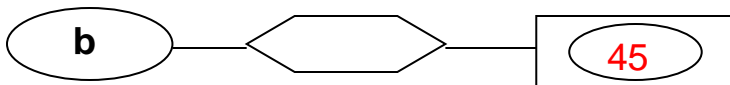
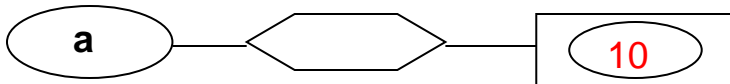
- Η πραγματική παράμετρος αποαναφοροποιείται και επιστρέφει μία τιμή, η οποία αντιγράφεται σε μια νέα θέση μνήμης, με την οποία συνδέεται το όνομα της τυπικής παραμέτρου. Δηλαδή, πρακτικά δημιουργείται μια νέα (τοπική) μεταβλητή.
- **C, C++, Pascal** (default)
- *Πλεονέκτημα*: Το ΥΠ μόνο διαβάζει την πραγματική παράμετρο, δεν έχει πρόσβαση για να την αλλάξει.
- *Μειονέκτημα*: Διπλασιασμός χρησιμοποιούμενης μνήμης.

Κλήση με Τιμή (2)

Σχηματικά:

Ορισμός: Procedure $P(x, y)$

Κλήση: $P(a, b)$



Κλήση με Αναφορά (1)

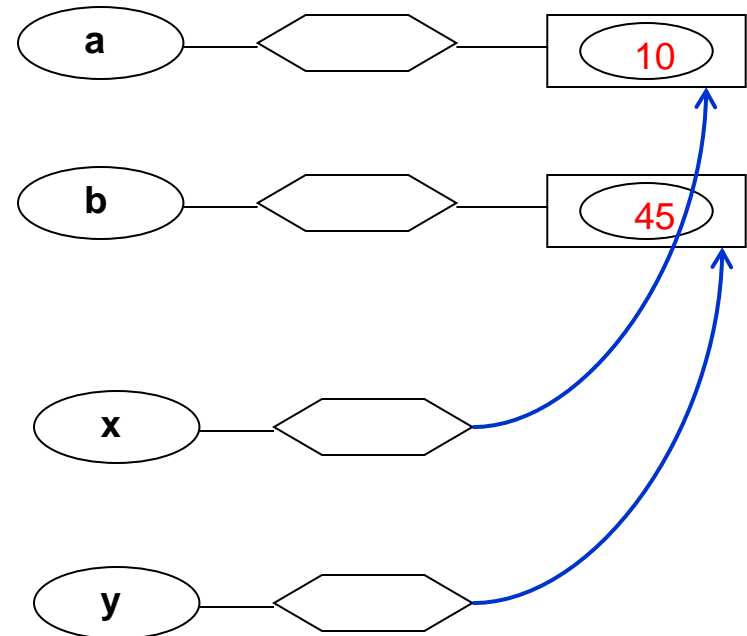
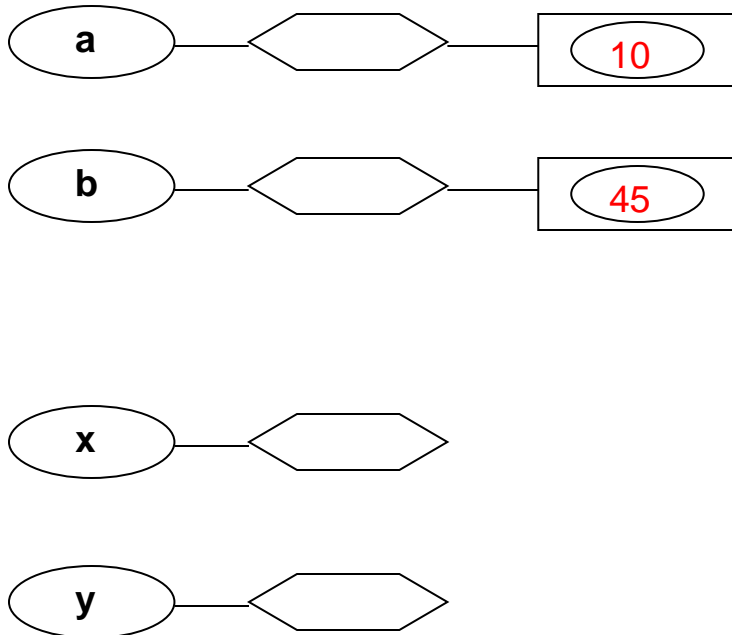
- Αν η πραγματική παράμετρος χρησιμοποιεί μνήμη (π.χ. είναι μεταβλητή), αυτή η μνήμη συνδέεται με την τυπική παράμετρο, όταν γίνεται κλήση του ΥΠ. Δηλαδή, πρακτικά η τυπική παράμετρος γίνεται pointer στην πραγματική παράμετρο.
- **FORTRAN, PL/1, Pascal** (με **VAR** στις τυπικές παραμέτρους)
- **C, C++** με χρήση pointers: `void A(int *f)` κλήση: `A(&x)`
- Κλήση με πραγματικές παραμέτρους σταθερές:
`P(var x)` Κλήση: `P(10)`
Αν αλλάζει η τιμή του x στο ΥΠ, η θέση μνήμης του λέγεται *ανώνυμη μεταβλητή*.
- *Πλεονεκτήματα*: Απόδοση, γρήγορη αλλαγή τιμής της πραγματικής παραμέτρου.

Κλήση με Αναφορά (2)

Σχηματικά:

Procedure P(VAR x, y)

Κλήση: P(a, b)



Κλήση με Τιμή - Αποτέλεσμα

- Όταν η πραγματική παράμετρος είναι μεταβλητή, αποαναφοροποιείται και η τιμή αντιγράφεται σε μια νέα θέση μνήμης, όπως στην Κλήση με Τιμή. Η θέση αυτή μνήμης, χρησιμοποιείται στο σώμα του ΥΠ. Κατά την έξοδο, η τιμή της τυπικής παραμέτρου αντιγράφεται στη θέση μνήμης της πραγματικής μεταβλητής.
- **ALGOL-W**

Κλήση με Όνομα

- Αφήνει τις πραγματικές παραμέτρους χωρίς να υπολογιστεί η τιμή τους, μέχρι το χρονικό **σημείο χρήσης** τους στο ΥΠ.
- Δηλαδή, οι *πραγματικές παράμετροι* αντιμετωπίζονται οι ίδιες σαν υποπρογράμματα χωρίς παραμέτρους (thunk), που εκτελούνται και υπολογίζεται η τιμή τους (για να δοθεί στην τυπική παράμετρο), με το τρέχον ΠΑ του προγράμματος ή ΥΠ το οποίο καλεί το τρέχον ΥΠ.
- Τότε, η τυπική παράμετρος συνδέεται με την πραγματική παράμετρο, όπως στην Κλήση με Αναφορά.
- Ο υπολογισμός της πραγματικής παραμέτρου, γίνεται εξ αρχής, κάθε φορά που χρησιμοποιείται η αντίστοιχη τυπική παράμετρος.
- **ALGOL-60, SIMULA** (επιλογή του χρήστη)

Παράδειγμα 1

MAIN

VAR **M**: integer;

C: array [1..10] of integer;

Procedure **R** (VAR **i**, **j**: integer)

begin

i = **i** + 1;

j = **j** + 1;

write (**i**, **j**);

end;

BEGIN

M := 2;

R (**M**, **C**[**M**]);

END.

Ερώτημα 1:

Με την εντολή **write(i,j)** τι θα τυπωθεί, αν έχουμε **Κλήση με Αναφορά;**

Απάντηση 1:

3, C [2] + 1

Ερώτημα 2:

Με την εντολή **write(i,j)** τι θα τυπωθεί, αν έχουμε **Κλήση με Όνομα;**

Απάντηση 2:

3, C [3] + 1

Ερώτημα 3:

Τι τιμή έχει το **C[3]** στο τέλος με

Κλήση με Όνομα;

Απάντηση 3:

C [3]_{αρχικό} + 1

Παράδειγμα 2

// C program to illustrate call by value

```
#include <stdio.h>
void swapx(int x, int y);
int main()
{
    int a = 10, b = 20;
    swapx(a, b);
    printf("a=%d b=%d\n", a, b);
    return 0;
}
void swapx(int x, int y)
{
    int t;
    t = x;
    x = y;
    y = t;
    printf("x=%d y=%d\n", x, y);
}
```

x=20 y=10
a=10 b=20

// C program to illustrate Call by Reference

```
#include <stdio.h>
void swapx(int*, int*);
int main()
{
    int a = 10, b = 20;
    swapx(&a, &b);
    printf("a=%d b=%d\n", a, b);
    return 0;
}
void swapx(int* x, int* y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
    printf("x=%d y=%d\n", *x, *y);
}
```

x=20 y=10
a=20 b=10

Διαφορές C, C++ και Java

- **Note** : In C, we use pointers to achieve call by reference. In C++, we can either use pointers or references to for pass by reference. In Java, primitive types are passed as values and non-primitive types are always references.
- **Java is Strictly Pass by Value!**

ΠΑΡΑΔΕΙΓΜΑ 3: Java is Strictly Pass by Value!

Primitive types:

```
public class Main
{
    public static void main(String[] args)
    {
        int x = 5;
        change(x);
        System.out.println(x);
    }
    public static void change(int x)
    {
        x = 10;
    }
}
```

Output: 5

How about objects or references?

The changes are not reflected back if we change the object itself to refer some other location or object.

Παράδειγμα 4: A Java program to show that references are also passed by value.

```
class Test
{
    int x;
    Test(int i) { x = i; }
    Test()    { x = 0; }
}

class Main
{
    public static void main(String[] args)
    {
        // t is a reference
        Test t = new Test(5);
        // Reference is passed and a copy of
reference
        // is created in change()
        change(t);
        // Old value of t.x is printed
        System.out.println(t.x);
    }
}

public static void change(Test t)
{
    // We changed reference to refer some other location
    // now any changes made to reference are not reflected
    // back in main
    t = new Test();
    t.x = 10;
}
```

Output: 5

Παράδειγμα 5: Changes are reflected back if we do not assign reference to a new location or object:

```
class Test
{
    int x;
    Test(int i) { x = i; }
    Test()    { x = 0; }
}

class Main
{
    public static void main(String[] args)
    {
        // t is a reference
        Test t = new Test(5);
        // Reference is passed and a copy of
reference
        // is created in change()
        change(t);
        // New value of t.x is printed
        System.out.println(t.x);
    }
}

// This change() doesn't change the reference, it only
// changes member of object referred by reference

public static void change(Test t)
{
    t.x = 10;
}
}
```

Output: 10

Παράδειγμα 6.1 (call-by-value)

```
program params;
  var i: integer;
  a: array[1..2] of integer;
  procedure p(x,y: integer);
  begin
    x := x + 1;
    i := i + 1;
    y := y + 1;
  end;
begin
  a[1] := 1;
  a[2] := 2;
  i := 1;
  p( a[i],a[i] );
  output( a[1],a[2] );
end.
```

Call by Value

x and **y** in **p** are **local** variables *initialized* with the actual parameters, while **i** is a **global** variable, so the call **p(a[i],a[i])** is equivalent to:

```
x := 1 /* The value of a[i] */
y := 1 /* The value of a[i] */
```

```
x := 2 /* x + 1 */
```

```
i := 2 /* i + 1 */
```

```
y := 2 /* y + 1 */
```

and at the end the values **1, 2** are printed since they are the values of **a[1], a[2] which weren't changed.**

Παράδειγμα 6.2 (call-by-reference)

```
program params;  
  var i: integer;  
  a: array[1..2] of integer;  
  procedure p(x,y: integer);  
  begin  
    x := x + 1;  
    i := i + 1;  
    y := y + 1;  
  end;  
begin  
  a[1] := 1;  
  a[2] := 2;  
  i := 1;  
  p( a[i],a[i] );  
  output( a[1],a[2] );  
end.
```

Call by Reference

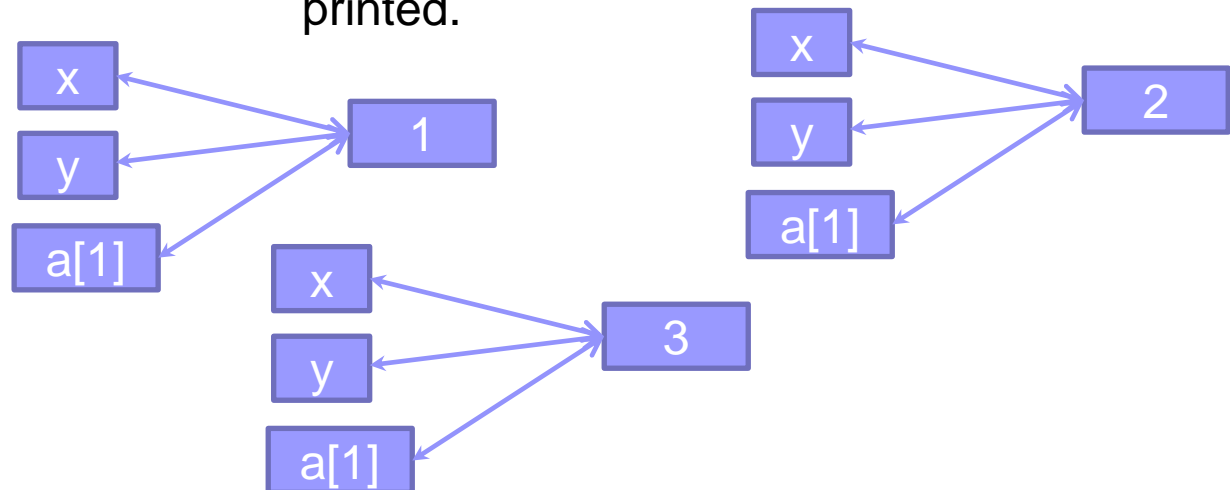
Both **x** and **y** in **p** are **alias** for **a[1]** and (again) **a[1]** (since **i = 1** when the procedure is called), so the call is equivalent to:

a[1] := 2 /* a[1] + 1 */

i := 2 /* i + 1 */

a[1] := 3 /* a[1] + 1 */

and at the end the values **3, 2** are printed.



Παράδειγμα 6.3 (call-by-name)

```
program params;
  var i: integer;
  a: array[1..2] of integer;
  procedure p(x,y: integer);
  begin
    x := x + 1;
    i := i + 1;
    y := y + 1;
  end;
begin
  a[1] := 1;
  a[2] := 2;
  i := 1;
  p( a[i],a[i] );
  output( a[1],a[2] );
end.
```

Call by Name is *equivalent* to Call by Reference when simple variables are passed as parameters, but it is *different* when you pass an expression that denotes a memory location, like a subscript.

In this case the actual parameter is re-evaluated each time it is encountered. So in this case, this is the effect of the call of `p(a[i],a[i])`:

`a[1] := 2` /* since `i = 1`, the result is equal to `a[1] + 1` */

`i := 2` /* `i + 1` */

`a[2] := 3` /* since `i` is now `2`, the result is equal to `a[2] + 1` */
and at the end the values **2, 3** are printed.

In practice the implementation calls an anonymous function (a “thunk”), each time it must evaluate a parameter.

Παράδειγμα 6.4 (call-by-value result)

program params;

var i: integer;

a: array[1..2] of integer;

procedure p(x,y: integer);

begin

x := x + 1;

i := i + 1;

y := y + 1;

end;

begin

a[1] := 1;

a[2] := 2;

i := 1;

p(a[i],a[i]);

output(a[1],a[2]);

end.

Call by Value Result:

x and y are initialized at the beginning of the procedure execution with the values of the actual parameters, and, at the end of the execution of the procedure, are copied back to the original variables addresses:

x := 1 /* The value of a[i] */

y := 1 /* The value of a[i] */

x := 2 /* x + 1 */

i := 2 /* i + 1 */

y := 2 /* y + 1 */

a[1] := 2 /* the value of x is copied back to a[1] */

a[1] := 2 /* the value of y is copied back to a[1] (not a[2]!) */

and at the end the values **2, 2** are printed.