Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

Διαχείριση με συναρτήσεις

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "mytypes.h"
Listelement_ptr createnewelement(char *, int);
Listelement_ptr findelementbyname(List, char []);
void reportlist( List const);
void addtolist(List *, Listelement_ptr);
int deleteelement(List *, Listelement_ptr);
int main(void) {
          /* Initialize */
          Listelement_ptr iterator=NULL, element_ptr, previous_ptr;
          List nameslist = NULL;
          char name[50];
          /* create elements and put them to list*/
          element_ptr = createnewelement("Ntina", 10);
          addtolist (&nameslist, element_ptr);
          element_ptr = createnewelement("Giannis", 5);
          addtolist( &nameslist, element_ptr);
          element_ptr = createnewelement("Maria", 7);
          addtolist( &nameslist, element_ptr);
          reportlist(nameslist);
          /* find, delete, report */
          do {
             printf("name: ");
             scanf("%s", name);
             element_ptr = findelementbyname(nameslist, name);
             if (element_ptr)  {
                       printf("found it. data: %d\n", element_ptr->age);
                       if (deleteelement(&nameslist, element_ptr) == 0)
                                          printf("...deleted.\n");
                       reportlist(nameslist);
                    }
          } while (nameslist!=NULL) ;
 return 0;
}
```

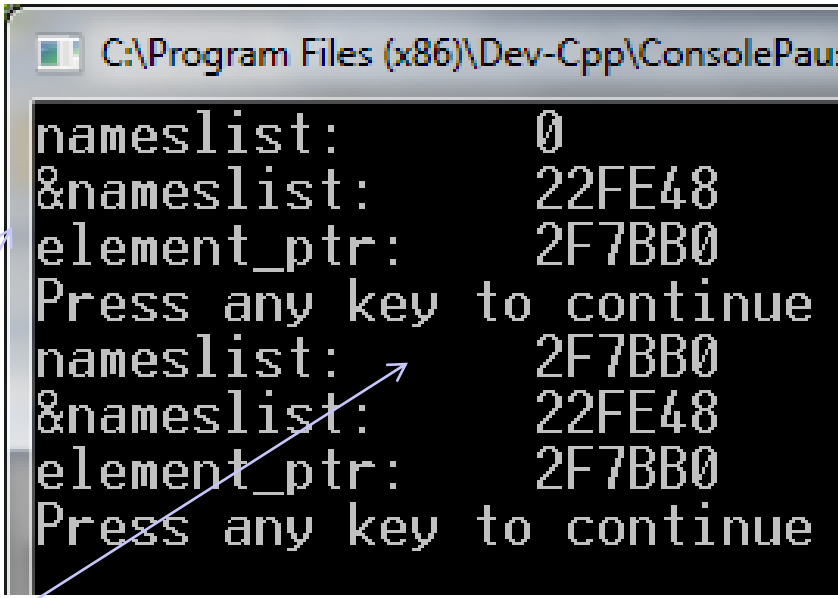
```
Listelement_ptr findelementbyname(List mylist, char name[]) {
        Listelement_ptr iterator;

        for (iterator = mylist; iterator != NULL; iterator = iterator -> next) {
                if (strcmp(iterator->name, name)==0 ) return iterator;
        }

        return NULL;
};
```

Λίστα ως παράμετρος με αναφορά

```
void addtolist (List *, Listelement_ptr );
List nameslist = NULL;
Listelement_ptr = element_ptr;
element_ptr = createnewelement("Ntina", 10);
addtolist (&nameslist, element_ptr);
```

Τιμές **πριν** την εκτέλεση της addtolist

Τιμές **μετά** την εκτέλεση της addtolist



- Η `nameslist` αποθηκεύεται στη θέση 22FE48.
- Η τιμή της `nameslist` αλλάζει μετά την κλήση της `addtolist`. (όχι η θέση της!)
- Τύπος της `nameslist`: `List` Τύπος της θέσης της: `List *`

Θέση της namelist (ή &nameslist)

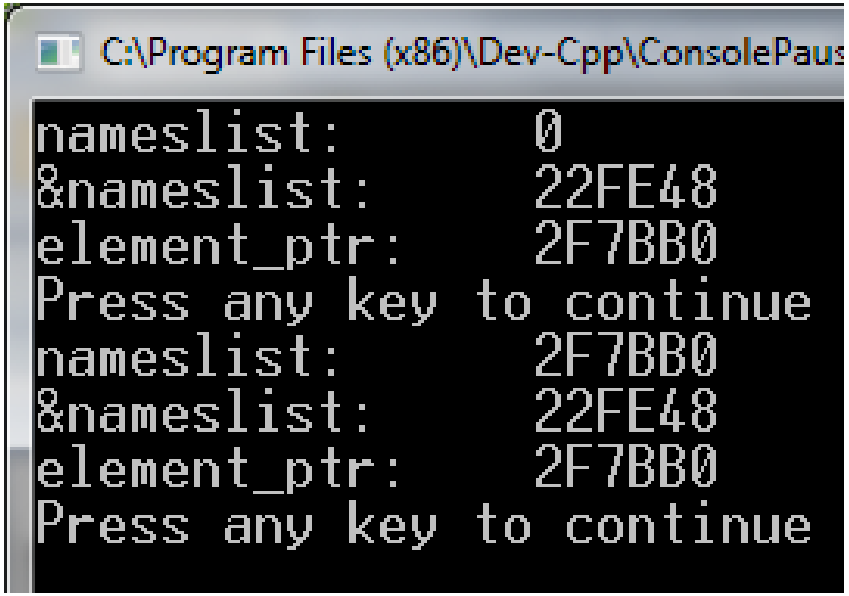
22FE48

Τιμή της namelist

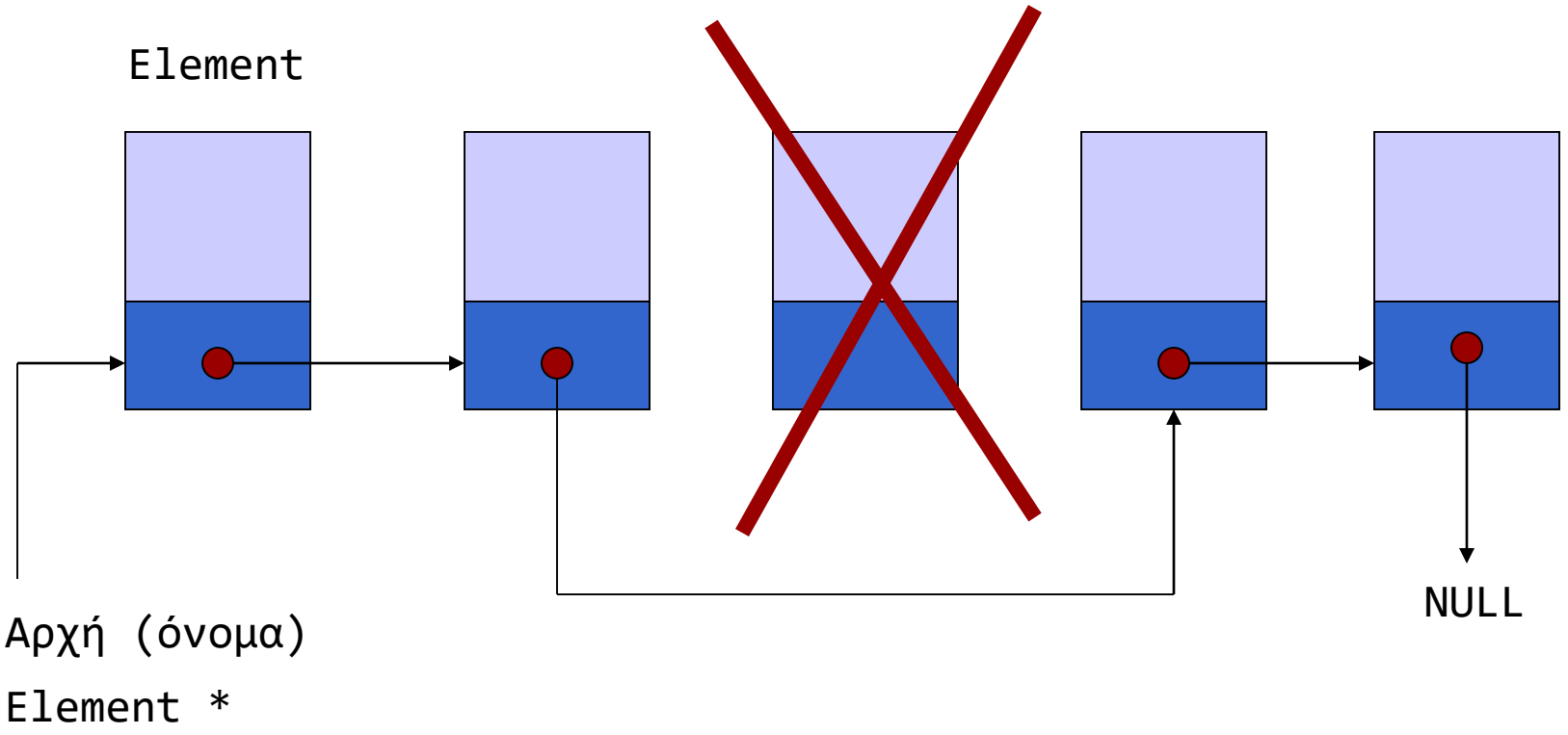
2F7BB0

Πριν την κλήση της addtolist

Μετά την κλήση της addtolist



Διαγραφή στοιχείου



Τι γίνεται με τη μνήμη την οποία το στοιχείο καταλάμβανε;

Διαγραφή με πρόβλεψη το στοιχείο να μην υπάρχει στη λίστα

```
#include <stdio.h>
#include <stdlib.h>
#include "mytypes.h"
int deleteelement(List *alist, Listelement_ptr  todelete_ptr) {
 Listelement_ptr  iterator = *alist;

if (todelete_ptr == NULL || *alist == NULL)
  return -1 ;
 if (todelete_ptr == *alist) {
   *alist = todelete_ptr->next ;
    free(todelete_ptr);
   }
 else {
    for(; iterator != NULL && iterator->next != todelete_ptr ;
                                        iterator = iterator->next) ;
    if (iterator!=NULL) {
          iterator->next = todelete_ptr->next;
          free(todelete_ptr);}
    else {
          printf("element not in list");
          return -1;
          }
     }
  return 0;
}
```

Όσο δείχνεις σε στοιχείο της λίστας (`iterator!=NULL`) ΚΑΙ το επόμενο στοιχείο δεν είναι το προς διαγραφή (`iterator->next != todelete_ptr`), πήγαινε στο επόμενο στοιχείο

Αν μετά το βρόχο το `iterator` δείχνει σε στοιχείο (`!=NULL`), αυτό είναι το ακριβώς προηγούμενο από το προς διαγραφή.

Διαγραφή με πρόβλεψη το στοιχείο να μην υπάρχει στη λίστα

```
#include <stdio.h>
#include <stdlib.h>
#include "mytypes.h"
int deleteelement(List *alist, Listelement_ptr  todelete_ptr) {
 Listelement_ptr  iterator = *alist;

if (todelete_ptr == NULL || *alist == NULL)
  return -1 ;
 if (todelete_ptr == *alist) { /* if required, delete first element*/
   *alist = todelete_ptr->next ;
    free(todelete_ptr);
   }
 else { /* find the element before the one to be deleted or
          continue until no more elements in list */
    for(; iterator != NULL && iterator->next != todelete_ptr ;
                                        iterator = iterator->next) ;
    if (iterator!=NULL) {
          iterator->next = todelete_ptr->next;
          free(todelete_ptr);}
    else {
          printf("element not in list");
          return -1;
          }
     }
  return 0;
}
```

Πρώτα χρησιμοποιώ το todelete_ptr
Μετά αποδεσμεύεται με free

Τι θα γίνει αν γράψουμε
`iterator->next!=todelete_ptr && iterator !=NULL`
αντί για
`iterator!=NULL && iterator->next!=todelete_ptr`

Χρήσιμα σημεία σε συναρτήσεις επεξεργασίας διασυνδεδεμένης λίστας (linked list)

- Η διασυνδεδεμένη λίστα ως παράμετρος
  - Αναφερόμαστε σε λίστα με τη διεύθυνση του πρώτου στοιχείου
  - Μερικές συναρτήσεις είναι δυνατό να αλλάξουν τη διεύθυνση του πρώτου στοιχείου (προσθέτουν, διαγράφουν, …)
  - Άλλες δεν αλλάζουν (εκτύπωση, καταμέτρηση, αναζήτηση…)
- Πρώτα χρησιμοποιώ τα στοιχεία, μετά διαγράφω
- Βρίσκω και καλύπτω ειδικές περιπτώσεις
  - Ενδεικτικά: άδεια λίστα, πρώτο στοιχείο, κενό στοιχείο ως είσοδος

Άλλο Παράδειγμα

- Παραγωγή ψευδοτυχαίων αριθμών και καταχώρηση σε απλά διασυνδεδεμένη λίστα της πληροφορίας
- Ζητείται στη λίστα αυτή κάθε αριθμός να καταχωρείται μόνο μία φορά.
  - ποιοι αριθμοί εμφανίστηκαν και
  - πόσες φορές ο καθένας.

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
            int num;
            int occur;
            struct node * next;
} ;
typedef struct node Node;
typedef Node * List;

Node *create(int num);
void report(List lst);
List update(List lst, int n);


 int main(void) {
            int s ;
            int i;
            List mylst = NULL;

            for (i = 0 ; i<10; i++) {
                        s = ((double) rand() / RAND_MAX ) * 5;
                        printf("before: %p ", (void *) mylst);
                        mylst = update(mylst, s);
                        printf("after: %p\n", (void *) mylst);
            }

            report(mylst);

            return EXIT_SUCCESS ;
 }
```

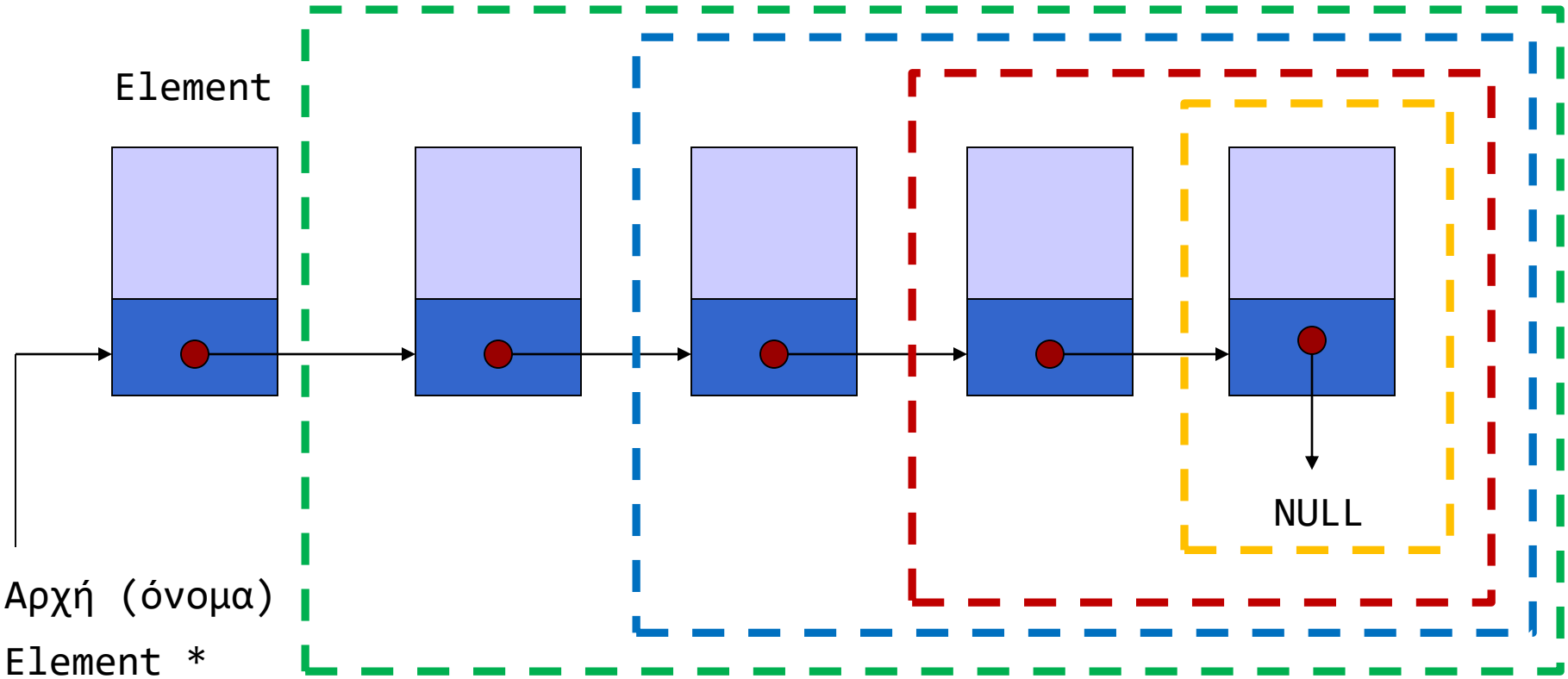
Συνάρτηση update χωρίς by-reference

```
List update (List mylst, int n) {
          List temp_list = mylst;
          Node * iter;

          if (mylst == NULL) {
                    temp_list = create(n);
                    return temp_list;
          }
          for (iter = temp_list;
                              iter->next != NULL;
                                        iter = iter -> next) {
                                                  if (iter -> num == n) {
                                                            (iter -> occur)++;
                                                            return temp_list;
                                                            }
                                                  }
          if (iter->num==n) {   (iter->occur)++;
                              return temp_list;
                              }

          iter->next = create(n);

          return temp_list;
}
```

```
Node * create (int n) {
            Node * tmp;

            tmp = malloc( sizeof (Node));
            tmp -> num = n;
            tmp -> occur = 1;
            tmp -> next = NULL;
            return tmp;
 }
```

```
void report(List lst) {
           Node * iter;

           for (iter = lst ; iter != NULL; iter = iter -> next){
                      printf("%d (%d):", iter->num, iter->occur);
           }
           return;
}
```

Αυτοαναφορική (self-referential) δομή



Ένας κόμβος δείχνει σε κόμβο ίδιου τύπου (ή NULL)

Άλλη ανάγνωση: Κάθε κόμβος δείχνει σε μικρότερη λίστα (ή NULL)

Αναδρομική αναζήτηση σε λίστα

```
#include <string.h>
#include "mytypes.h"

Listelement_ptr recfind(List alist, char name[]) {
if (alist == NULL)
   return NULL;
else
       if (!strcmp(alist->name, name))
          return alist;
       else
          return recfind(alist->next, name);
}
```

Απλοποίηση κώδικα

- Αξιοποίηση αυτοαναφορικής δομής
- Αναδρομική διαχείριση λίστας

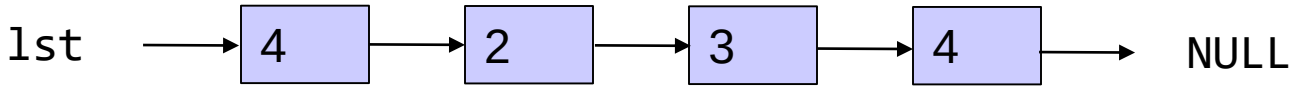
```
struct node {
        int value ;
        struct node * next;
};
typedef struct node Node;
typedef struct node * Node_ptr;
typedef struct node * List;
```

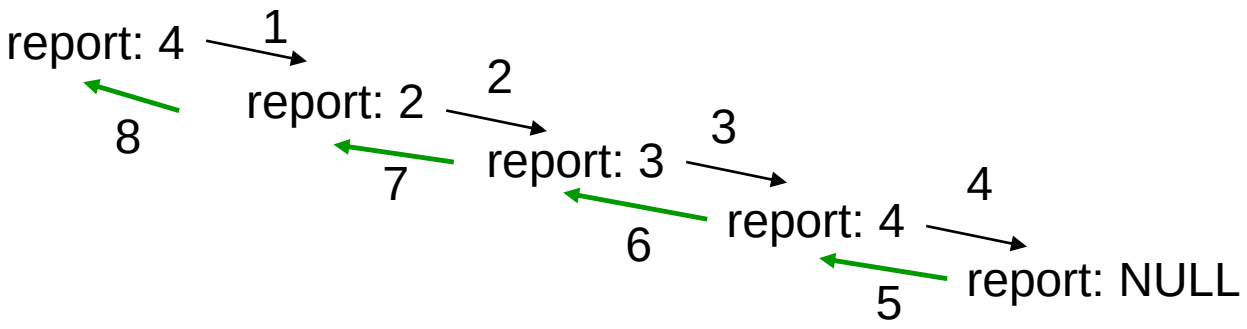
```
Node_ptr createnode(int a);
void report(List lst) ;
void append ( List * list_ptr, Node_ptr node_ref);
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "mylst.h"
#define N 10

int main(void ) {

        int i ;
        int temp ;
        List mylst = NULL;
        Node_ptr node_ref;

        srand(0);

        for (i=0; i<N; i++) {
                temp = rand() % 10 ;
                node_ref = createnode(temp);
                append( &mylst, node_ref) ;
        }

        report(mylst);

        return EXIT_SUCCESS;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "mylst.h"

Node_ptr createnode(int a) {
        Node_ptr new_node_ptr;

        new_node_ptr = malloc( sizeof (Node) ) ;
        new_node_ptr -> value = a;
        new_node_ptr -> next = NULL;

        return new_node_ptr;

}
```

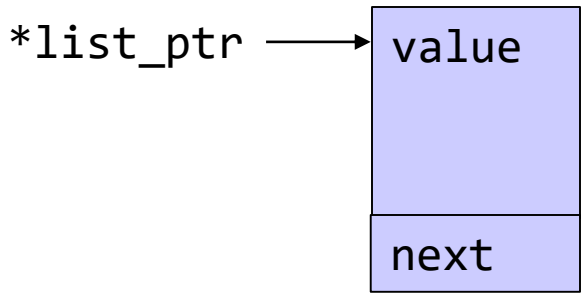
```
void report(List lst) {

        if (lst == NULL) return;

        printf("%d ", lst -> value);
        report ( lst->next);

        return ;
}
```





Να θυμηθούμε το stack, recursion depth limits

```
void append ( List * list_ptr, Node_ptr node_ref) {

        if ( *list_ptr == NULL) {
                *list_ptr = node_ref;
                return ;
        };

        append( &( (*list_ptr)->next ), node_ref );

        return ;
}
```



(*list_ptr)->next είναι lvalue

Βρίσκεται στη διεύθυνση &((*list_ptr)->next)

Αναδρομικές υλοποιήσεις

Υπολογισμός Παραγοντικού

$$3! = 1 \times 2 \times 3 = 6$$

$$n! = \prod_{i=1}^{n} i$$

$$0! = 1$$

$$n! = n(n-1)!$$

$$0! = 1$$

αναδρομικός τύπος

Αναδρομικότητα στη C

```c
#include <stdio.h>
int f(int);

int main ( void ) {
      int a = 3;
      printf ("%d\n", f(a));
      return 0;
      }

int f(int n) {
      int temp;
      if (n==0)
            temp = 1 ;
      else
            temp = n * f(n - 1);
      return temp;
      }
```

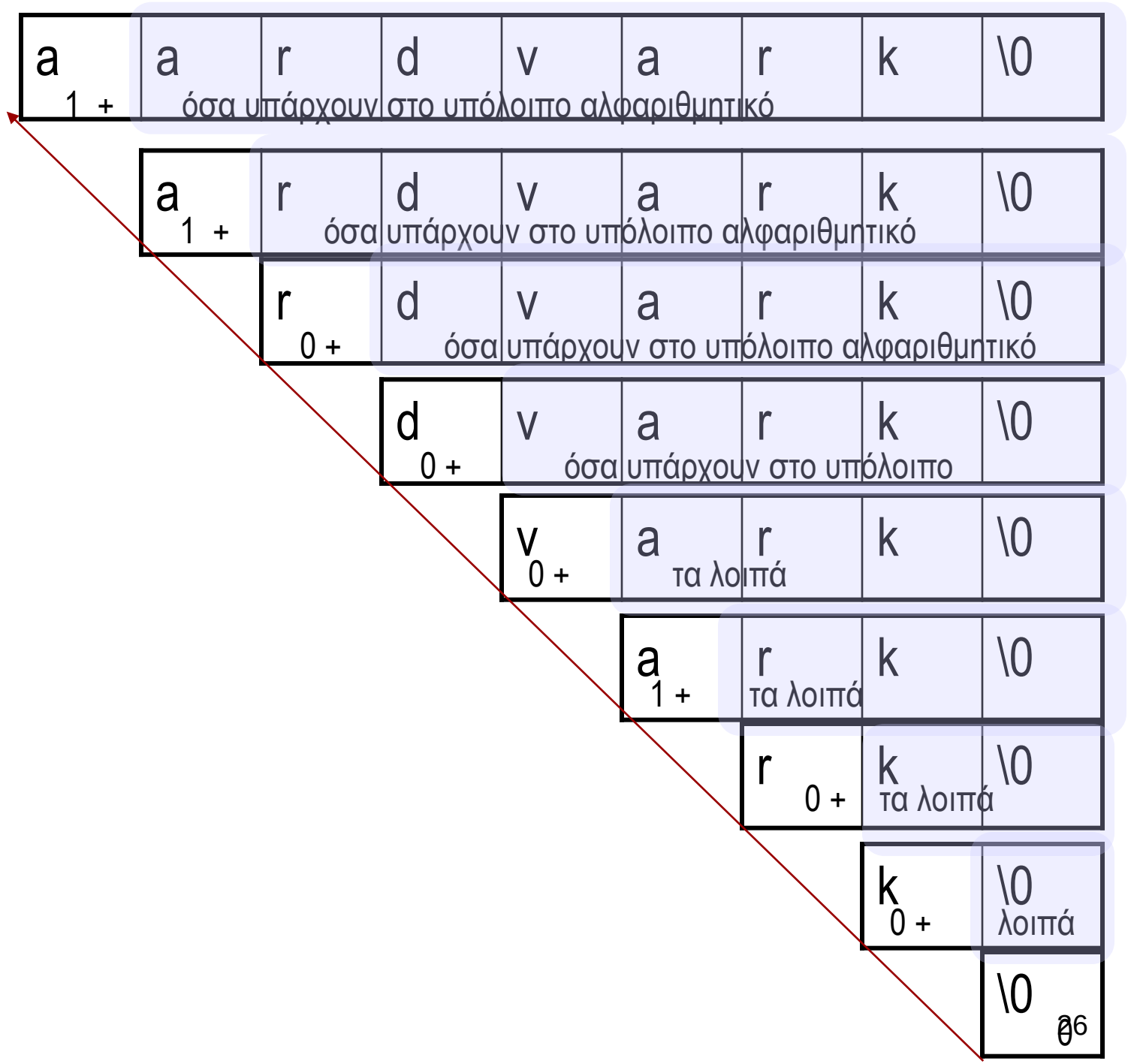
Χρειάζεται συνθήκη τερματισμού.

Η συνάρτηση f( ) καλεί τον εαυτό της στον **ορισμό** της.

Λύση χωρίς αναδρομικότητα

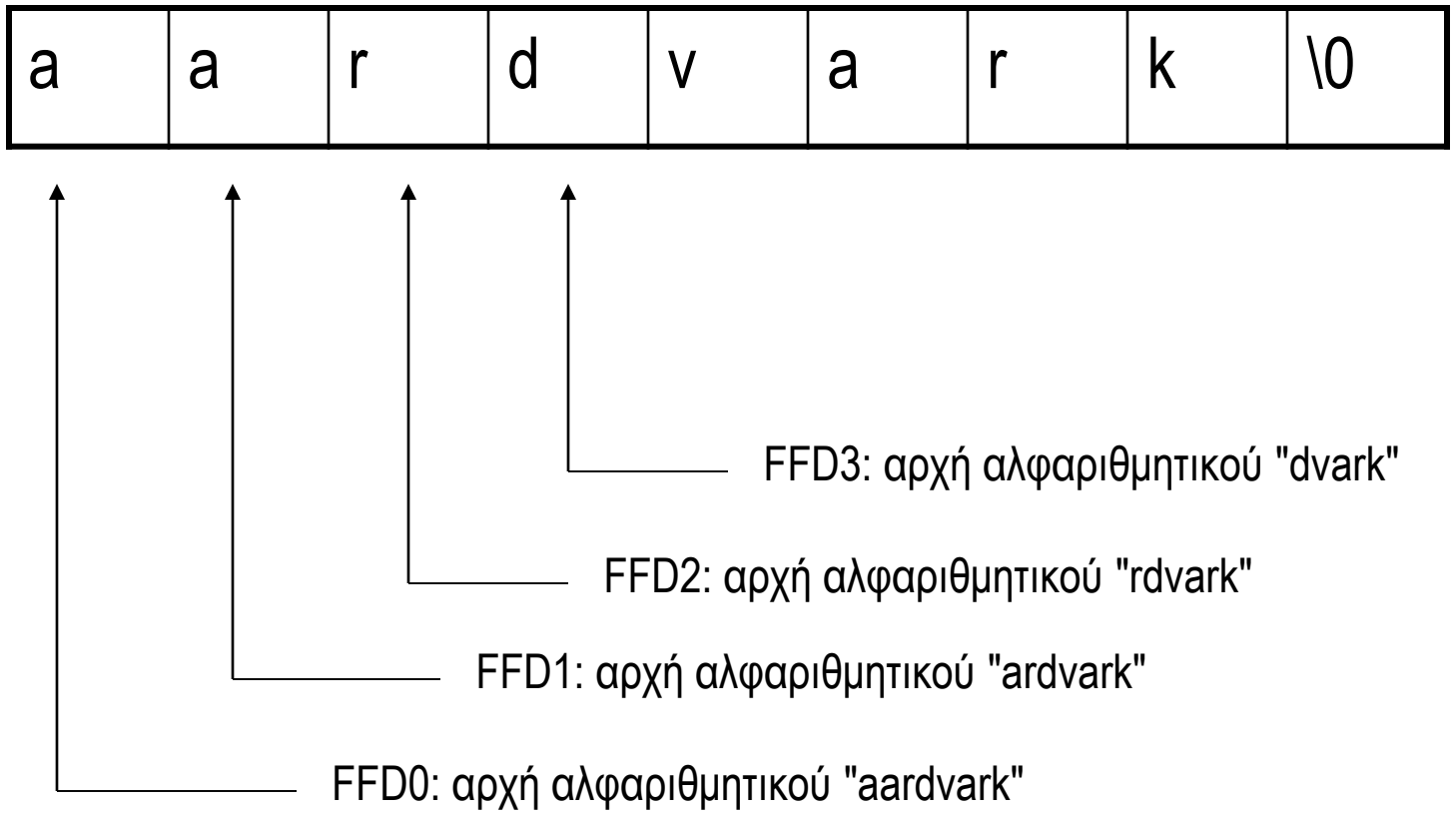
```
int f(int n) {
   int temp = 1 , i;

   for (i=1;i<=n;i++)
              temp *= i;

   return temp;
   }
```

```
int f(int n) {
   int temp = 1 , i;

   for(i=1;i<=n;temp*=(i++));

   return temp;
   }
```


a
a r d v a r k \0
1 +
όσα υπάρχουν στο υπόλοιπο αλφαριθμητικό
a r d v a r k \0
1 +
όσα υπάρχουν στο υπόλοιπο αλφαριθμητικό
r d v a r k \0
0 +
όσα υπάρχουν στο υπόλοιπο αλφαριθμητικό
d v a r k \0
0 +
όσα υπάρχουν στο υπόλοιπο
v a r k \0
0 +
τα λοιπά
a r k \0
1 +
τα λοιπά
r k \0
0 +
τα λοιπά
k \0
0 +
λοιπά
\0
0 +

Να γραφεί συνάρτηση που να μετρά το πλήθος εμφανίσεων ενός χαρακτήρα c σε ένα αλφαριθμητικό

- Αν υπάρχει αλφαριθμητικό:
  - Ο αριθμός των c στο αλφαριθμητικό είναι
    - αν ο πρώτος χαρακτήρας είναι c, είναι 1 + ο αριθμός των c που υπάρχουν στο υπόλοιπο αλφαριθμητικό
    - αν ο πρώτος χαρακτήρας δεν είναι c, είναι **μόνο** ο αριθμός των c που υπάρχουν στο υπόλοιπο αλφαριθμητικό.
- Ποια είναι η συνθήκη τερματισμού;
- Τι σημαίνει υπόλοιπο αλφαριθμητικό;


a
a
r
d
v
a
r
k
\0
FFD3: αρχή αλφαριθμητικού "dvark"
FFD2: αρχή αλφαριθμητικού "rdvark"
FFD1: αρχή αλφαριθμητικού "ardvark"
FFD0: αρχή αλφαριθμητικού "aardvark"

Αναδρομική υλοποίηση της συνάρτησης

```
int countchar(char *s, char c) {
  if (s[0] != '\0')
  if (s[0] == c)
      return 1+countchar(++s,c);
  else
      return countchar(++s, c);
 else
    return 0;
}
```

αν **δεν είναι** το τέλος του αλφαριθμητικού

τότε αν **ο πρώτος χαρακτήρας είναι** c

το αποτέλεσμα είναι 1 + όσα c υπάρχουν στο υπόλοιπο αλφαριθμητικό

αλλιώς το αποτέλεσμα είναι μόνο όσα 'a' υπάρχουν στο υπόλοιπο αλφαριθμητικό

αν **είναι** το τέλος του αλφαριθμητικού, δεν υπάρχουν c, άρα επιστρέφει 0

Υλοποίηση με τον τελεστή ? :

```
int countchar(char *s, char c) {
if (s[0])
 return(s[0]==c)?1+countchar(++s,c):countchar(++s,c);
else
 return 0;
}
```

Χρήση τελεστή ? : αντί για **if else**

Άλλη ισοδύναμη υλοποίηση (χωρίς εσωτερικό **if else**)

```
int countchar(char *s, char c) {
  if (s[0])
   return ((s[0] == c) +countchar(++s, c)) ;
 else
    return 0;
}
```

Αντικατάσταση του **if else** με τελεστή ? :

```
int countchar(char *s, char c) {
  return  s[0]?(s[0] == c) + countchar(++s,c):0;
}
```

```
int countchar(char *s, char c) {
  return  (*s)?(*s == c) + countchar(s+1,c):0;
}
```

Είναι αυτό αναδρομή;

```
#include <stdio.h>
int max(int, int);
int main ( ) {
  int a = 1, b = 2 , c= 3, d;
  d = max(a, max(a, b));
  printf("%d", d);
  return 0;
  }
```

```c
#include <stdio.h>

int findmax(int a[], int size);

int main( ) {
        int data[] = {1,3,7,1,4,3};
        printf("max:%d\n", findmax(data, 6));
        return 0;
}

int findmax(int a[], int size) {
        int b;
        if (size==1) {
                return a[0];
        }
        b = findmax(a+1, size-1);
        if (a[0]>b)
                return a[0];
        else
                return b;
}
```

```
int getmin (int a[], int size) {
        int i , temp;
        temp = a[0];

        for (i=1; i < size; i++) {
                if ( temp>a[i] ) temp = a[i];
                }

        return temp;
}
```

```
int getmax (int a[], int size) {
        int i , temp;
        temp = a[0];

        for (i=1; i < size; i++) {
                if ( temp<a[i] ) temp = a[i];
                }

        return temp;
}
```

```
int getmin (int a[], int size) {
 int i , temp;
 temp = a[0];

 for (i=1; i < size; i++) {
    if ( temp>a[i] ) temp = a[i];
    }

 return temp;
}
```

```
int getmax (int a[], int size) {
  int i , temp;
  temp = a[0];

  for (i=1; i < size; i++) {
    if ( temp<a[i] ) temp = a[i];
    }

  return temp;
}
```

Όνομα συνάρτησης ως τιμή

- Οι παρενθέσεις ως τελεστής
  - Δήλωση/πρότυπο συνάρτησης

```
int f (int );
```

  - Κλήση συνάρτησης

```
int a ;
a = f(5) ;
```

- Το όνομα συνάρτησης μόνο του $\Rightarrow$ διεύθυνση
- Μπορώ να δηλώσω σχετική **μεταβλητή**

Δείκτης σε συνάρτηση

```
/* δήλωση δείκτη σε
 * ακέραιο
 */
int *intptr;

/* πρότυπο συνάρτησης
 * που επιστρέφει
 * δείκτη σε ακέραιο
 */
int *f();

/* δήλωση δείκτη
 * σε συνάρτηση */
int (*f)() ;
```

```
int function1(void);
int function2(void);
int main ( ) {
   int a = 0;
   int (*f)( );

/* some code here */
   if (!a)
        f = function1;
   else
        f= function2;

   (*f)();

   return 0;
}
```

```c
#include <stdio.h>

int map (int a[], int size, int (*f)(int, int));
int more(int a, int b);
int less(int a, int b);

int main(void) {

int data[7]= {-1, 2, 1, 0, 5, 7, -3};

 printf("max: %2d\n", map(data, 7, more));
 printf("max: %2d\n", map(data, 7, less));

 return 0;
}
```

```c
int less (int a, int b) {
          return a > b;
}


int more (int a, int b) {
          return a < b;
}


int map (int a[], int size, int (*f)(int, int)) {
 int i , temp;
 temp = a[0];

 for (i=1; i < size; i++)
          if ( f(temp, a[i])) temp = a[i];

 return temp;

}
```

Πίνακας Δεικτών σε Συνάρτηση - Αρχικοποίηση

```c
#include <stdlib.h>
#include <stdio.h>
double one( double );
double two( double );

int main (void ) {
        double (*f[2])(double) = {one, two} ;
        double x = 3.47, y;
        int i ;

        do {
                printf("select function:\t");
                scanf("%d", &i);
                y = (*f[i-1])( x) ;
                printf("result %g\n", y);
        } while (1) ;
        return EXIT_SUCCESS;
}

double one(double x) { return (x + 1.0) ; }
double two(double x ) { return (x + 2.0) ; }
```