Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

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;
      }
```

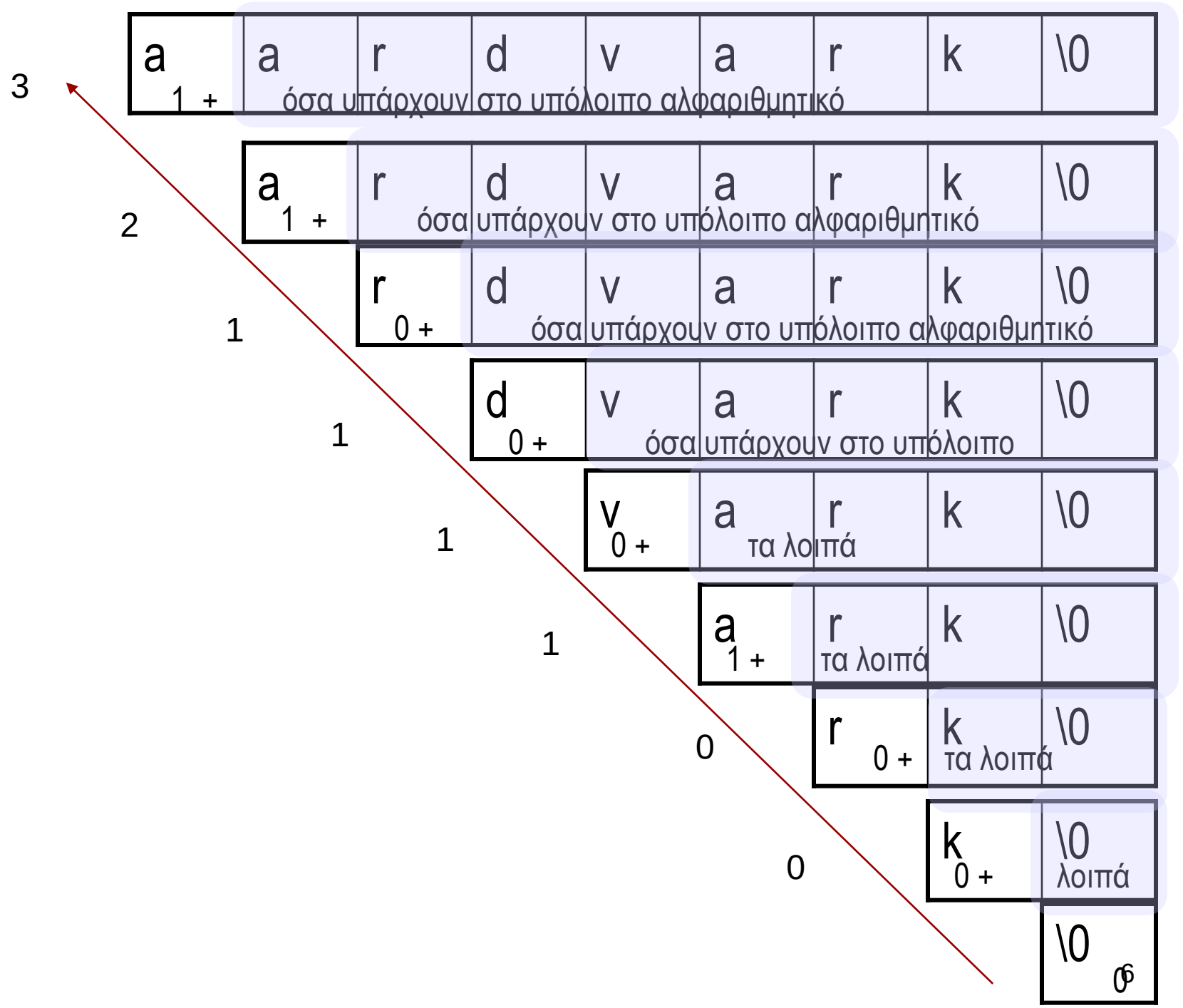
Χρειάζεται συνθήκη τερματισμού.

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

Λύση χωρίς αναδρομικότητα

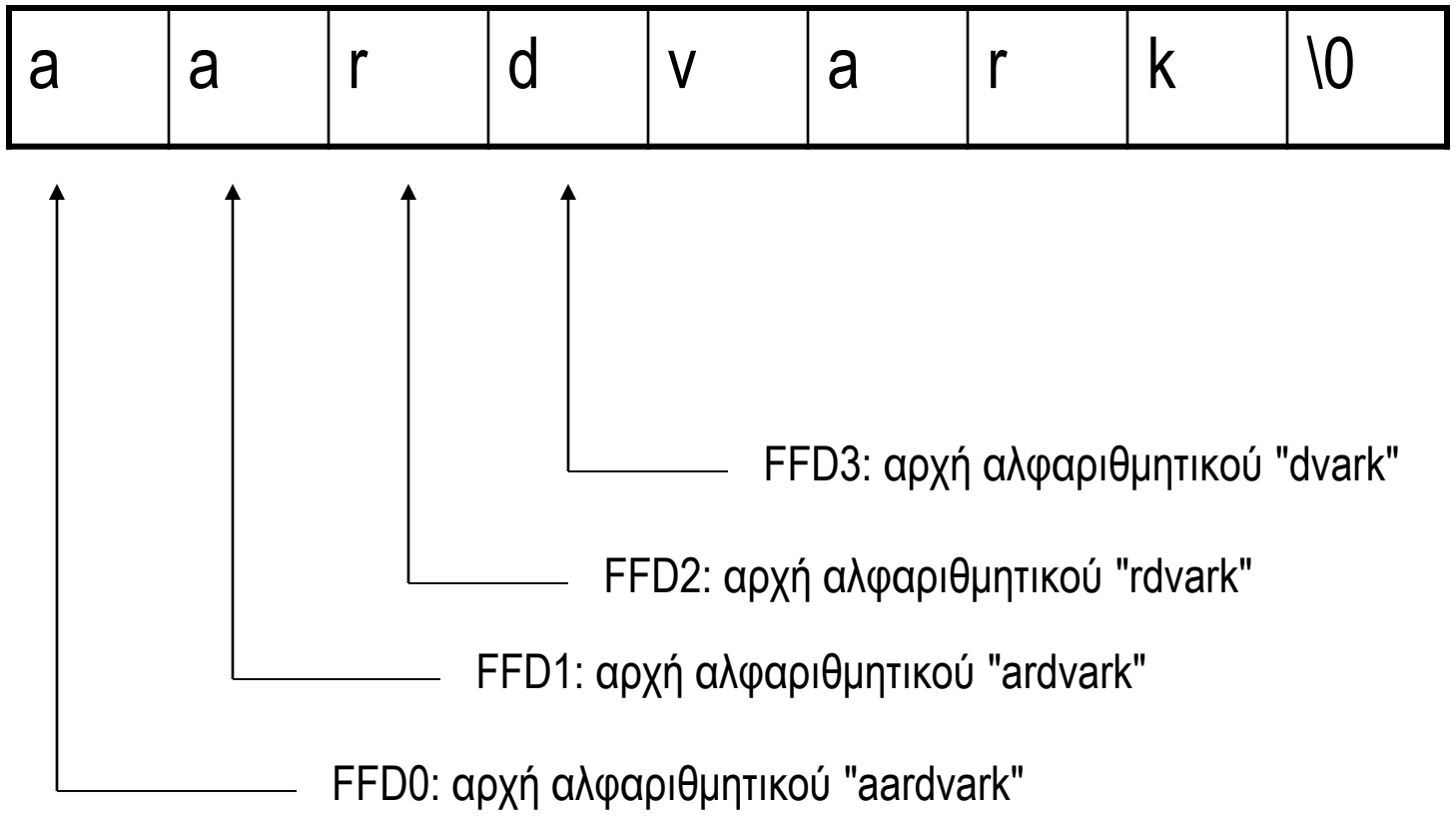
```
int f(int n) {
   int temp = 1 , i;

   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
3
a r d v a r k \0
1 + όσα υπάρχουν στο υπόλοιπο αλφαριθμητικό
2
r d v a r k \0
0 + όσα υπάρχουν στο υπόλοιπο αλφαριθμητικό
1
d v a r k \0
0 + όσα υπάρχουν στο υπόλοιπο
1
v a r k \0
0 + τα λοιπά
1
a r k \0
1 + τα λοιπά
0
r k \0
0 + τα λοιπά
0
k \0
0 + λοιπά
\0
0
6

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;

}
```

Βρες το μέγιστο αναδρομικά

```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;
}
```

Το μέγιστο είναι το πρώτο στοιχείο αν είναι μεγαλύτερο από το μέγιστο των υπολοίπων, διαφορετικά είναι το μέγιστο των υπολοίπων.

```
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("min: %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

```
#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);
                if (i>=1 && i <=2  ) {
                        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) ;
                }
```

Με **typedef** για ευανάγνωστο κώδικα

```c
#include <stdio.h>

typedef int (*Callback)(int, int);

int map (int a[], int size, Callback f);
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("min: %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, Callback f) {
 int i , temp;
 temp = a[0];

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

 return temp;

}
```

Με typedef

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

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

        do {
                printf("select function:\t");
                scanf("%d", &i);
                if (i>=1 && i <=2  ) {
                        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) ;
                }
```