Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς

Πολλαπλές επιλογές: switch

```
switch (έκφραση)  {
   case τιμή1: εντολές ; break;
   case τιμή2: εντολές ; break;
   case τιμή3: εντολές ; break;
   /* … */
   default: εντολές ; break;
   }
```

case τιμή1: λειτουργεί ως label

break: μεταφέρει τον έλεγχο εκτός του switch () {}

τι γίνεται χωρίς **break?**

```c
#include <stdio.h>
int getchoice (void) ;
void start (void) ;
void stop (void);

int main (){

    int userchoice ;

    while ((userchoice = getchoice()) != 3){
     switch (userchoice) {
        case 1: start() ;
                break;
        case 2: stop();
                break;
        default: break;
     }
   }

   return 0;
}
```

```c
int getchoice (void ) {
 int choice ;

 printf("1: start\n2: stop\n3:quit\n");
 printf("enter choice:\n");
 scanf("%d", &choice);

 return choice;
}

void start (void) {
    printf("Start...");
}

void stop (void) {
    printf("Stop...");
}
```

Πίνακες στη C

- Δεσμεύουν ***συνεχή χώρο*** στη μνήμη

- Οι ακόλουθες δηλώσεις οδηγούν τον compiler να δημιουργήσει διαφορετική assembly.

```
int a[32];

int s = 32;
int a[s];
```

Παράδειγμα

```
#include <stdio.h>
int main ( ) {
   int i;
   float temp[3]  = { -2.1, 5.5, 10.1};

   for (i=0; i< 3; i = i + 1)
      printf("\ttemp[%d]: %f\n", i, temp[i]);

   return 0;
}
```

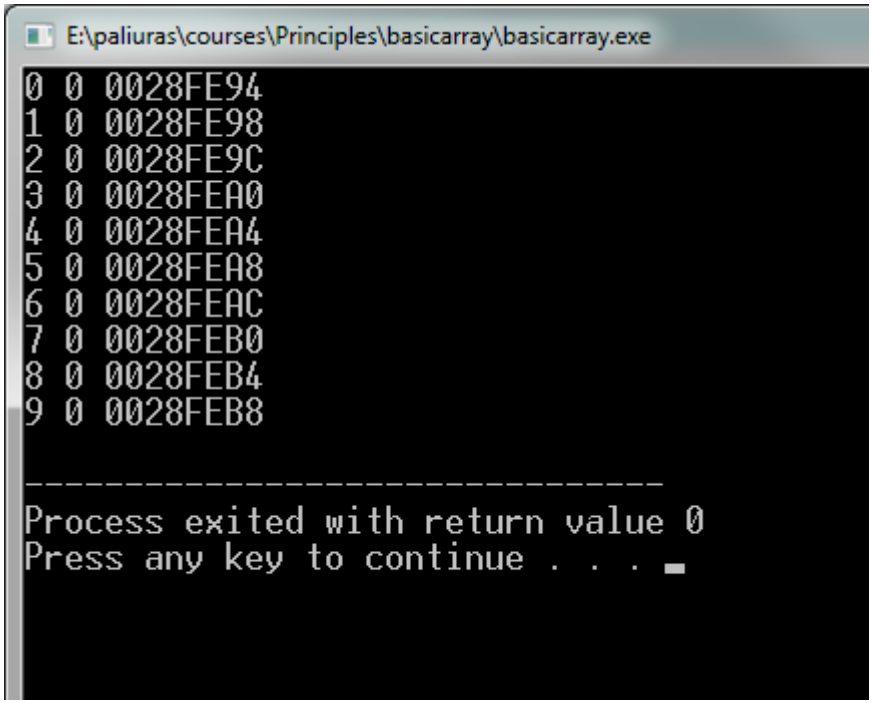
```
printf("\ttemp[%d]: %+6.2f\n", i, temp[i]);
```

cygwin

Πίνακες

- Συλλογή μεταβλητών **ίδιου τύπου**, οι οποίες αποθηκεύονται σε διαδοχικές θέσεις μνήμης.
- **float** temperature[31];
  - *δήλωση* πίνακα μεταβλητών **float,** 31 στοιχείων
  - temperature[**0**] είναι το **πρώτο** στοιχείο,
  - temperature[**1**] είναι το **δεύτερο** στοιχείο,
  - ...
  - temperature[**30**] είναι το **τριακοστό πρώτο** στοιχείο,
  - temperature είναι **η διεύθυνση του πρώτου στοιχείου**
    - temperature είναι το ίδιο με &temperature[0]

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

int main() {

        int i ;
        int data[N] = {0};

        for (i=0; i< 10; i++)
                printf("%d %d %p\n", i, data[i], &data[i]);

        return 0;
}
```



0	1	2	3	4	5	6	7	8	9

Πίνακες δύο (ή περισσότερων) διαστάσεων

- `int a[3][3] ;`
- `int a[3][3] = {{1,2,3}, {3,2,1}, {1,1,1}};`

1	2	3
3	2	1
1	1	1

```c
#include <stdio.h>
#define N 3

int main ( ) {
int i, j;

int a[N][N] = {{1,2,3}, {3,2,1}, {1,1,1}};

for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
            printf("%d ",a[i][j]);
            }
    printf("\n");
}

return 0;
}
```

Αποθήκευση στη μνήμη - πίνακας μιας διάστασης

```
int a[3] = {10,20,30};
```

Διεύθυνση	Στοιχείο	Περιεχόμενα
1000	a[0]	10
1004	a[1]	20
1008	a[2]	30

Αποθήκευση στη μνήμη πίνακας δύο διαστάσεων

a[1] σημαίνει δεύτερη γραμμή
a[1][2] σημαίνει τρίτο στοιχείο δεύτερης γραμμής

1	2	3
3	2	1
1	1	1

Διεύθυνση	Στοιχείο
1000	1
1004	2
1008	3
100C	3
1010	2
1014	1
1018	1
101C	1
1020	1

```c
#include <stdio.h>
#define N 3

int main ( ) {
int i, j;

int a[N][N] = {{1,2,3}, {3,2,1}, {1,1,1}};
int *b = &a[0][0];

for (i=0; i< N; i++) {
    for (j=0; j< N; j++) {
            printf("%d ",a[i][j]);
            }
    printf("\n");
    }

for (i=0; i< N*N; i++) {
    printf("%d ", *(b+i));
    }

return 0;
}
```

Συνάρτηση της βασικής βιβλιοθήκης scanf ( )

```
int number;
char ch;
scanf("%d", &number);
scanf("%c", &ch);
```

%d θα διαβάσει ακέραιο

%c θα διαβάσει χαρακτήρα

τελεστής διεύθυνσης (&): Επιστρέφει τη διεύθυνση της θέσης μνήμης η οποία αντιστοιχεί στη μεταβλητή που ακολουθεί

Παράδειγμα

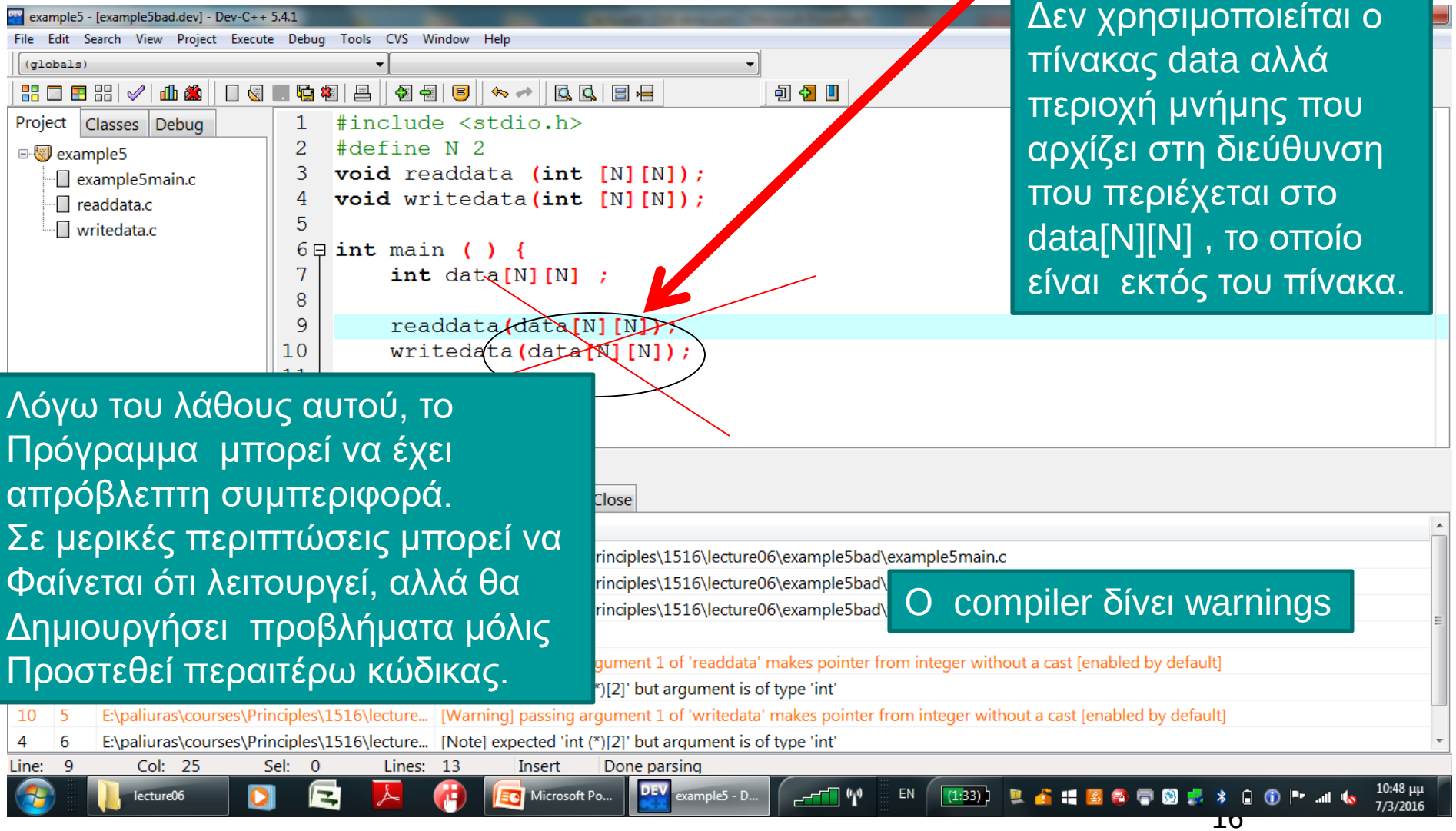
```
#define N 2
#include <stdio.h>
int main ( ) {
   int data[N][N] ;
   int i, j ;

   for (i =0 ; i < N ; i++)
    for ( j = 0 ; j < N ; j ++) {
      printf ("element (%d,%d)?\t", i, j);
      scanf("%d", &data[i][j]);
      }

   for (i =0 ; i < N ; i++) {
    for ( j = 0 ; j < N ; j ++) {
        printf ("%d\t", data[i][j]);
        }
     printf("\n");
   }
 return 0;
}
```

```c
#define N 2
#include <stdio.h>
void readdata(int [N][N]);
void writedata(int [N][N]);

int main ( ) {
   int data[N][N] ;

   readdata(data) ;
   writedata(data);

   return 0;
}
```

```c
void readdata(int a[N][N]) {
int i,j;
for (i =0 ; i < N ; i++)
   for ( j = 0 ; j < N ; j ++) {
     printf ("element (%d,%d)?\t", i, j);
     scanf("%d", &a[i][j]);
     }
}

void writedata(int b[N][N]) {
int i,j;
for (i =0 ; i < N ; i++) {
   for ( j = 0 ; j < N ; j ++)
        printf ("%d\t", b[i][j]);
   printf("\n");
 }
}
```

Καλύτερα!

ΠΡΟΣΟΧΗ!!! ΤΕΡΑΣΤΙΟ ΛΑΘΟΣ!!!



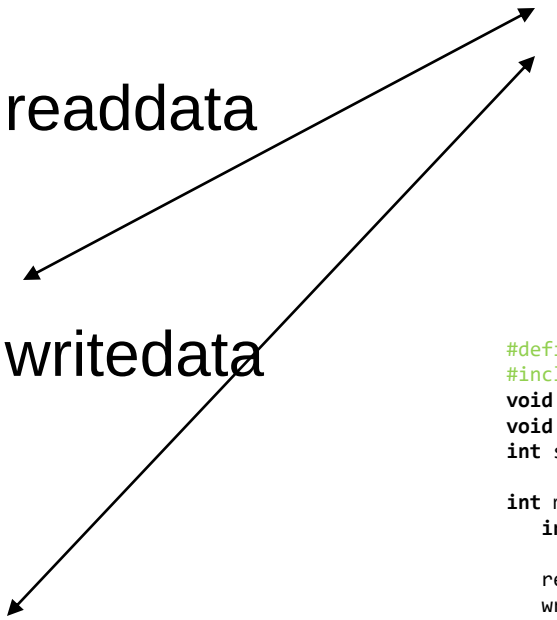
```c
#define N 2
#include <stdio.h>
void readdata(int a[N][N]);
void writedata(int b[N][N]);
int sumdata(int x[N][N]);

int main ( ) {
   int data[N][N] ;

   readdata(data) ;
   writedata(data);

   printf("The sum is: %d\n",
   sumdata(data));

return 0;
}
```

```c
int sumdata(int x[N][N]) {
   int i, j;
   int sum = 0;

   for (i=0; i<N; i++)
        for (j=0; j<N; j++)
                sum += x[i][j];

return sum;
}
```

- Συνάρτηση main
  - Πίνακας data
  - Καλείται η συνάρτηση readdata
    - Όρισμα a
  - Καλείται η συνάρτηση writedata
    - Όρισμα b

Στη μνήμη υπάρχει μόνο ένας πίνακας!

Πίνακας δεδομένων data



```
#define N 2
#include <stdio.h>
void readdata(int a[N][N]);
void writedata(int b[N][N]);
int sumdata(int x[N][N]);

int main ( ) {
   int data[N][N] ;

   readdata(data) ;
   writedata(data);

   printf("The sum is: %d\n",
                sumdata(data));

return 0;
}
```

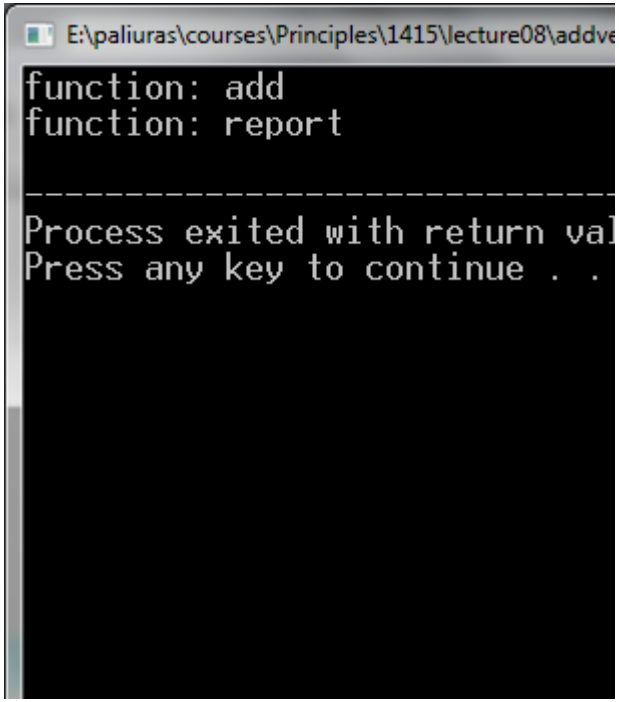
Άσκηση

- Να γραφεί πρόγραμμα που να αθροίζει δύο διανύσματα Ν στοιχείων σε ISO C90 χρησιμοποιώντας πίνακες ακεραίων.

Δομημένη λεκτική περιγραφή

- Δήλωσε/αρχικοποίησε πίνακες a, b, c
- Άθροισε πίνακες α, b με αποτέλεσμα στον c
  - συνάρτηση add
- Εμφάνισε τον πίνακα c
  - συνάρτηση report

```c
#include <stdio.h>
#define N 5
void add (const int [], const int [], int []);
void report (const int []);

int main() {
        int a[N] = {1, 2, 3, 4, 5};
        int b[N] = {6, 7, 8, 9, 0};
        int c[N];

        add(a, b, c);
        report (c);

        return 0;
}
```



- Πλήρης main
- Κενές add, report

```c
void add(const int a[N], const int b[N], int c[N] ) {
        printf("add vectors\n");
        return ;
}

void report (const int c[N]) {
        printf("report\n");
        return ;
}
```

```
void add(const int a[N], const int b[N], int c[N] ) {
        int i;

        for (i=0; i<N; i++)
                c[i] = b[i] + a[i];

        return ;
}

void report (const int c[N]) {
        int i;

        for (i=0; i<N; i++)
                printf("%d ", c[i]);

        printf("\n");
        return ;
}
```

- Πλήρης υλοποίηση συναρτήσεων add, report

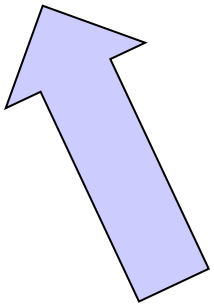
Μια προγραμματιστική τεχνική

- ***Εξασφαλίζουμε*** ότι μια συνάρτηση μπορεί να αλλάξει τιμές πίνακα ***μόνο αν*** αναλυτικά το επιτρέψουμε.
- Εφαρμογή της αρχής ***ελαχίστου δικαιώματος*** *(principle of least privilege).*
- Χρήση τύπου **`const int [ ]`**

Πίνακες ως είσοδοι και έξοδοι

Είσοδοι: δεν επιτρέπεται στη συνάρτηση να αλλάξει τις τιμές στοιχείων πινάκων **const int** []

```
void add(const int a[N], const int b[N], int c[N] ) {
        int i;

        for (i=0; i<N; i++)
                c[i] = b[i] + a[i];

        return ;
}
```



Έξοδος : η συνάρτηση έχει δικαίωμα να Αλλάξει τα στοιχεία του πίνακα **int** []

```
void report (const int c[N]) {
        int i;

        for (i=0; i<N; printf("%d ", c[i++]));

        printf("\n");
        return ;
}
```

- Άλλη υλοποίηση της report:
  - Η τρίτη έκφραση στο (κενό) **for** τυπώνει και αυξάνει το μετρητή με postfix increment

ΣΥΝΗΘΗ ΛΑΘΗ

```
/* Σωστό !!! */
int main() {
        int a[N] = {1, 2, 3, 4, 5};
        int b[N] = {6, 7, 8, 9, 0};
        int c[N];

        add(a, b, c);
        report (c);

        return 0;
}
```

```
c = add(a, b); /* Λάθος: Το c δεν μπορεί να αλλάξει, είναι η
                  διεύθυνση του πρώτου στοιχείου του πίνακα! */

add(a[], b[], c[]); /* Λάθος: Εδώ είναι  syntax error. Μόνο σε
        δήλωση μπορεί να παραληφθεί μια (και μόνο μία) διάσταση (η
        τελευταία). */

add(a[N], b[N], c[N]); /* Λάθος: Η τιμή ενός ακεραίου (έξω από τους
πίνακες) μεταφράζεται σε διεύθυνση!!!  Warning: pointer from integer
without a cast */
```

Βασικός τύπος **char**

```c
#include <stdio.h>

int main() {

        char a;

        a = 'g';

        printf("this is a char: %c\n", a);

        return 0;
}
```

Διαφορετική σημασία απλών / διπλών εισαγωγικών στη C

- Διαφορετικά από python
- Απλά εισαγωγικά → ένας χαρακτήρας

```
char ch;
ch = 'a';
```

- Διπλά εισαγωγικά → ακολουθία χαρακτήρων που τερματίζεται με την αξία 0

```
char name[10] = "myname";
printf("stay safe\n");
printf("hello");
```

- Χαρακτήρας και κώδικας ascii

Αλφαριθμητικά (strings)

πρόκειται για **πίνακες χαρακτήρων**:

- `char name[30];`

- αρχικοποίηση με
  `char name[30] = "abcd";`
  το οποίο ισοδυναμεί με

```
name[0] = 'a';
name[1] = 'b';
name[2] = 'c';
name[3] = 'd';
name[4] = 0 ; /* δηλώνει το τέλος ενός
αλφαριθμητικού */
```

Ανάγνωση και εκτύπωση αλφαριθμητικού

- `char str[N_MAX + 1];`
- `scanf ("%s", str );`
- `printf ("%s\n", str );`

- `%s` → αντιστοιχεί σε αλφαριθμητικό
- `str[0]` είναι ο πρώτος χαρακτήρας
- `str` είναι η **διεύθυνση του πρώτου στοιχείου**
  - str είναι το **ίδιο** με &str[0]
  - ισχύει για κάθε τύπο πίνακα

Παράδειγμα

- Το σύστημα ζητά από το χρήστη το όνομά του και τυπώνει "hello" ακολουθούμενο από το όνομα του χρήστη.

Υλοποίηση σε C

```
#include <stdio.h>
#include <string.h>
#define N 10

int main (  ) {

 char username[N];

 printf("Please enter user name: ");
 scanf("%s", username);

 printf("Hello, %s\n", username);
 printf("Your name is %d letters long", strlen(username));


return  0;
}
```

το όνομα του πίνακα είναι η διεύθυνση του πρώτου στοιχείου (⇒ δεν βάζουμε &)

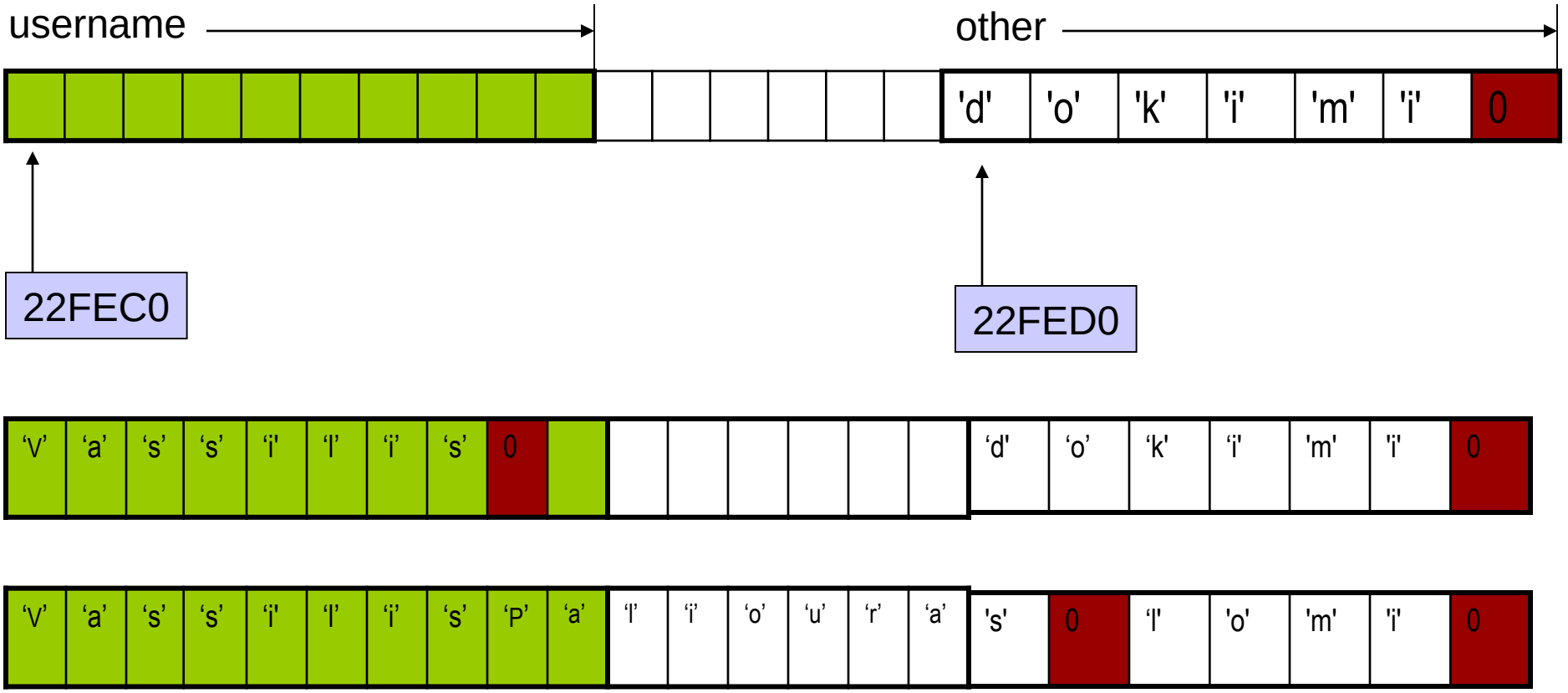
συνάρτηση βιβλιοθήκης strlen()

Υπάρχουν όρια;

```
#include <stdio.h>
#include <string.h>
#define N 10

int main (  ) {

 char other[ ] = "dokimi";
 char username[N];

 printf("Please enter user name: ");
 scanf("%s", username);

 printf("Hello, %s\n", username);
 printf("Your name is %d letters long stored at %X\n", strlen(username), username);

printf("Value of other: %s at %X", other, other);

return 0;
}
```

Ναι, αλλά δεν γίνεται έλεγχος...

Σχηματικά η μνήμη – buffer overflow



Καταστρέφονται τα περιεχόμενα της other!

Επεξεργασία ανά χαρακτήρα

```c
#include <stdio.h>
int main() {
   char string1[ 20 ], string2[] = "string literal";
   int i;

   printf(" Enter a string: ");
   scanf( "%s", string1 );
   printf( "string1 is: %s\nstring2: is %s\n"
           "string1 with spaces between characters is:\n",
           string1, string2 );

   for ( i = 0; string1[ i ] != '\0'; i++ ) {
      printf( "%c ", string1[ i ] );
   }
   printf( "\n" );
   return 0;
}
```

Μια τεχνική

- **Εξασφαλίζουμε** ότι μια συνάρτηση μπορεί να αλλάξει τιμές πίνακα μόνο αν αναλυτικά το επιτρέψουμε.
- Εφαρμογή της αρχής *ελαχίστου δικαιώματος.*

```c
#include <stdio.h>
void DisplayName(char []);

int main ( ) {
char astring[10] = "Hello";

DisplayName(astring);
DisplayName(astring);

return 0;
}

void DisplayName(char x[]) {
int i;
for (i=0; x[i]!=0 ; i++)
   printf("%c", x[i]);
x[0]='h';
printf("\n");
}
```

```c
#include <stdio.h>
void DisplayName(const char []);

int main ( ) {
char astring[10] = "Hello";

DisplayName(astring);
DisplayName(astring);

return 0;
}

void DisplayName(const char x[]) {
int i;
for (i=0; x[i]!=0 ; i++)
   printf("%c", x[i]);
x[0]='h';
printf("\n");
}
```

Error: assignment of read-only location

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

int main(void) {
   char alphabet[27]; // 26 letters plus trailing zero
   char c;

   for (c='A'; c<='Z'; c++)
        alphabet[c-'A'] = c;

   alphabet[c-'A'] = 0;

   printf("%s", alphabet);

   return 0;
}
```

Δείκτες (Pointers)

- **Δείκτης**: μεταβλητή στην οποία αποθηκεύουμε **διεύθυνση** θέσης μνήμης.
  - δείχνει ***που*** είναι αποθηκευμένα δεδομένα
- **Δήλωση Δείκτη**

  `<τύπος>  *<όνομα δείκτη>;`
- ΠΡΟΣΟΧΗ: Το όνομα πίνακα **είναι** διεύθυνση, αλλά **δεν είναι** μεταβλητή!!!

Παράδειγμα δήλωσης δείκτη

- `char *ch_ptr;`
- Η μεταβλητή ch_ptr περιέχει ***διεύθυνση μνήμης*** στην οποία είναι αποθηκευμένο δεδομένο τύπου χαρακτήρα.
- `char ch;`
- Η μεταβλητή ch έχει ως αξία χαρακτήρα.

- Μπορούμε να δηλώσουμε δείκτες σε δεδομένα διαφόρων τύπων
  - Βασικών
  - Κατασκευασμένων

Παράδειγμα χρήσης δείκτη

```
void main ( ) {
char ch = 'a', ch2;
char *ch_ptr ;

ch_ptr = &ch ;
ch2 = *ch_ptr ;

printf ("%c", ch2);

}
```

Δήλωση δείκτη σε χαρακτήρα

* ⇒ **Περιεχόμενα της θέσης** στην οποία δείχνει ο δείκτης

Πίνακες και δείκτες

- `int arr[10], n ;`
- `*(arr + n)` ⇔ `arr[n]`
- `arr + n` ⇔ `&arr[n]`
- χρησιμοποιούμε δείκτες για να περάσουμε ως όρισμα σε συνάρτηση πίνακες
  - ακριβέστερα: σε ποια διεύθυνση μνήμης βρίσκεται το πρώτο στοιχείο του πίνακα.

Συναρτήσεις βασικής βιβλιοθήκης για αλφαριθμητικά

- πρότυπα στο <string.h>
- **char** *strcpy (**char** *, **const char** *) ;
- **int** strcmp (**const char** *, **const char** *) ;
- **char** *strcat (**char** *, **const char** *) ;
- **char** *strchr (**const char** *, **char**) ;
- size_t strlen (**const char** *) ;
- …

Παράδειγμα

- Διάβασε ένα αλφαριθμητικό
- Μέτρησε πόσες φορές περιλαμβάνει τον χαρακτήρα ‘a’
- Τύπωσε το αποτέλεσμα.

```
void readstring(char *) ;
int countA(char *);
void printresult( int ) ;
```

Η main ( ) του παραδ

```
#define N 50
#include <stdio.h>
void readstring(char *) ;
int countA(char *);
void printresult( int ) ;
```

```
main ( ) {
 char astring[N];
 int acount ;

 readstring (astring) ;

 acount = countA(astring);

 printresult(acount) ;
}
```

```
void readstring(char *s ) {
        printf ("alpharithmitiko? ");
        scanf("%s", s);
}
```

```
void printresult ( int a) {
        printf("The result is %d\n", a);
}
```

Υλοποίηση της **int** countA(**char** *);

```
int countA(char *s) {
int count = 0 ;
int i = 0;
  while ( s[i] != 0 ) {
     if (s[i] == 'a')
        count ++;
     i ++ ;
     }
  return count ;
}
```

γιατί το μηδέν δηλώνει τέλος του αλφαριθμητικού

αν ο τρέχων χαρακτήρας είναι ίσος με ‘a’, αύξησε το μετρητή count κατά ένα

προχώρησε στον επόμενο χαρακτήρα του αλφαριθμητικού

Πρόθεμα και Επίθεμα (prefix και postfix)

- i++; /*postfix */
- ++ i; /* prefix */
- i = 0;
- myprint(i++);
- i = 0;
- myprint(++i);

Η myprint ( ) καλείται με διαφορετικό όρισμα (διαφορετική τιμή) στις δύο περιπτώσεις!

Πρόβλημα

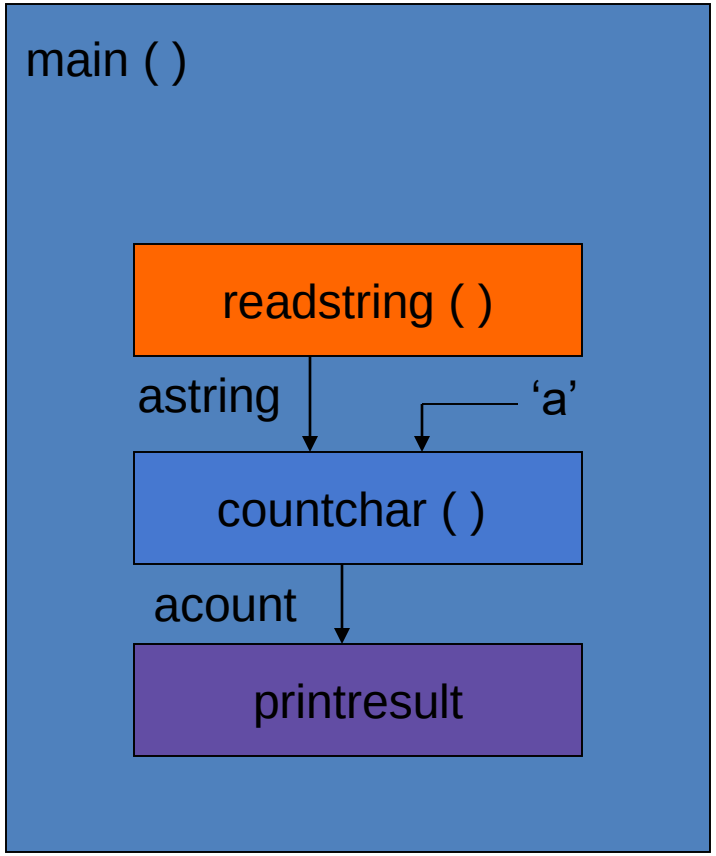
- Πώς θα γράφαμε πρότυπο και τον ορισμό συνάρτησης η οποία θα δέχεται ως ορίσματα:
  α) το αλφαριθμητικό και
  β) τον χαρακτήρα για τον οποίο γίνεται ο έλεγχος.
- Πρότυπο αυτής:

  int countchar(**char** *, **char**);
- Παράδειγμα κλήσης

  acount = countchar(astring, 'a');
- Τι πλεονεκτήματα έχει να γράφουμε γενικότερο κώδικα;

Χρήση της countchar(**char ***, **char**)

```
main ( ) {
 char astring[N];
 int acount ;

 readstring (astring) ;

 acount = countchar(astring, ‘a’);

 printresult(acount) ;
}
```



Πιθανές υλοποιήσεις

```
int countA(char *s, char ch)
    {
int count = 0 ;
int i = 0;
  while ( s[i] != 0 ) {
    if (s[i] == ch)
       count ++;
    i ++ ;
    }
  return count ;
}
```

```
int countA(char *s, char ch)
    {
int count = 0 ;
int i ;
  for (i=0; s[i] != 0; i++)
    if (s[i] == ch)
              count ++;
   return count ;
}
```

int countchar(**char** *, **char**);

```
int countchar(char *s, char c) {
int count = 0 ;
int i = 0;
  while ( s[i] ) {
                count += (s[i] == c);
              i ++ ;
              }
  return count ;
}
```

```
int countchar(char *s, char c) {
int count = 0 ;
int i = 0;                          postfix notation
  while ( s[i] )                          ↓
                count += (s[i++] == c);
return count ;
}
```

int countchar(**char** *, **char**);

```
int countchar(char *s, char c) {
int count = 0 ;
int i = 0;
  while ( s[i] ) {
                count += (s[i] == c);
              i ++ ;
              }
  return count ;
}
```

```
int countchar(char *s, char c) {
int count = 0 ;
int i = 0;                          postfix notation
  while ( s[i] )                          ↓
                count += (s[i++] == c);
return count ;
}
```

```
int countchar(char *s, char c) {
int count = 0 ;
int i ;

  for( i = 0; s[i]; count += (s[i++] == c));

return count ;
}
```

```
int countchar(char *s, char c) {
int count = 0 ;
int i = 0;

  for( ; s[i]; count += (s[i++] == c));

return count ;
}
```

Μηχανισμοί κλήσης συναρτήσεων

Τι θα τυπωθεί;

```
#include <stdio.h>

int myfun(int, int);

main ( ) {
int a = 3, b = 3;

myfun(a, b);
printf("main: a:%d b:%d\n", a,
    b);

}
```

```
int myfun(int a, int b) {
a++;
b++;
printf("myfun: a:%d b:%d\n",
    a, b);
return 0;
}
```

Κλήση συνάρτησης κατ' αναφορά (call by reference)

- πρότυπο: **void** swap(**int** *a, **int** *b);

- κλήση: swap(&value1, &value2);

1. Η συνάρτηση επενεργεί **απευθείας στις θέσεις μνήμης** των μεταβλητών που χρησιμοποιούνται ως πραγματικά ορίσματα.
2. **Δεν** δημιουργούνται τοπικά αντίγραφα των δεδομένων.
   - Ισχύει για τις διευθύνσεις;
3. Στη C, ουσιαστικά, υλοποιείται ως κατ' αξία πέρασμα διευθύνσεων.

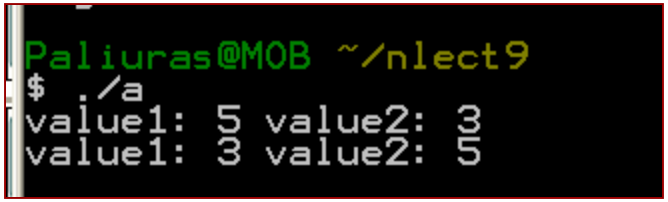
Ορισμός συνάρτησης swap( )

```
void swap(int *a, int *b) {
    int temp ;
    temp = *b ;
    *b = *a ;
    *a = temp ;
}
```

Παράδειγμα χρήσης κλήσης κατ΄αναφορά

```
#include <stdio.h>
void swap (int *, int *);

main ( ) {
 int value1 = 5;
 int value2 = 3;

 printf("value1: %d value2: %d\n", value1, value2);

 swap (&value1, &value2) ;

 printf("value1: %d value2: %d\n", value1, value2);
}
```

```
void swap(int *a , int *b) {
int temp;
 temp = *a ;
 *a = *b ;
 *b = temp ;
}
```



Μερικές διαφορές

- Μηχανισμός κλήσης κατ' αξία
  - δημιουργούνται τοπικά αντίγραφα των ορισμάτων, τα οποία χρησιμοποιεί η συνάρτηση
  - Αν τα ορίσματα έχουν μήκος πολλών bytes, επιβαρύνεται η εκτέλεση.
  - Η συνάρτηση δεν μπορεί να αλλάξει τις τιμές ορισμάτων στο σημείο κλήσης.
- Μηχανισμός κλήσης με αναφορά
  - δεν δημιουργούνται τοπικά αντίγραφα, η συνάρτηση ενημερώνεται για τη θέση μνήμης στην οποία είναι αποθηκευμένο ένα όρισμα.
    - μπορεί να είναι ταχύτερο
  - είναι δυνατόν να αλλάξουν οι τιμές ορισμάτων στο σημείο κλήσης.
    - χρειάζεται προσοχή, μπορεί να γίνει εκ παραδρομής...

```
void function (void);
void function1 (void);
main () {
        function1();
        function ( ) ;
        function ( ) ;
        function ( ) ;
}
void function1() {
int j = 0;
}
void function ( ) {
int i;
        i++ ;
        printf("%d\n", i);
}
```

Διάρκεια μεταβλητής

Η τοπική μεταβλητή

i δεν αρχικοποιείται!

Δουλεύει «σωστά»(;;;)

Γιατί;

```
void function1 (void);
void function2 (void);
void function(void);
main ( ) {
        function();
        function1( ) ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
}
void function() {
int k = 0;
}
void function1( ) {
int i;
        i++ ;
        printf("f1:%d\n", i);
}
void function2( ) {
int j;
        j++ ;
        printf("f2:%d\n", j);
}
```

Διάρκεια μεταβλητής (2)

Αν η κλήση της function1( ) γίνεται εναλλάξ με την function2( ), η συμπεριφορά αλλάζει...

Εμφανίζονται εξαρτήσεις (είναι δυνατόν να ...)

```
void function1 (void);
void function2 (void);

main ( ) {
        function1( ) ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
}
void function1 () {
int i = 0;
        i++ ;
        printf("f1:%d\n", i);
}
void function2 () {
int j = 0;
        j++ ;
        printf("f2:%d\n", j);
}
```

Λύση;;;

Αρχικοποιώντας τις μεταβλητές

κάθε φορά που καλείται η συνάρτηση,

οι τοπικές μεταβλητές μηδενίζονται κάθε

φορά που καλείται η συνάρτηση.

```
void function1 (void);
void function2 (void);

main ( ) {
        function1( )  ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
        function1( ) ;
        function2( ) ;
}
void function1 ( ) {
static int i = 0;
        i++ ;
        printf("f1:%d\n", i);
}
void function2 ( ) {
static int j = 0;
        j++ ;
        printf("f2:%d\n", j);
}
```

Η λέξη κλειδί **static** σε δηλώσεις τοπικών μεταβλητών

static: ο χώρος μνήμης της μεταβλητής δεν αποδεσμεύεται όταν ολοκληρωθεί μια εκτέλεση της συνάρτησης.

Άλλη χρήση της **static**: Ορισμός εμβέλειας αρχείου

- αρχείο πηγαίου κώδικα: code1.c

```
int func (int);
void report(void);

main () {
int i ;

for ( i=0; i< 5; i++)
    printf("%d\n", func(i));

report();
}
```

- αρχείο πηγαίου κώδικα: code2.c

```
static int sum = 0;

int func(int k) {
int i;
for (i=0;i<5; i++)
    sum += k ;  /* sum = sum + k */
return 2 * k;
}
void report() {
    printf("%d\n", sum);
}
```