

Διαδικαστικός Προγραμματισμός

Βασίλης Παλιουράς
paliuras@ece.upatras.gr

Βασικός τύπος char και ASCII

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

```
int main() {
```

```
    char a;
```

```
    a = 'g';
```

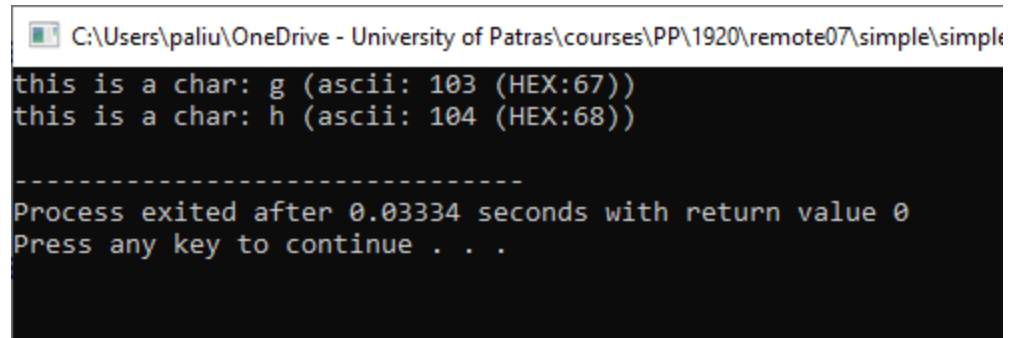
```
    printf("this is a char: %c (ascii: %d (HEX:%X))\n", a, a, a);
```

```
    a = '\x68';
```

```
    printf("this is a char: %c (ascii: %d (HEX:%X))\n", a, a, a);
```

```
    return 0;
```

```
}
```



```
C:\Users\paliu\OneDrive - University of Patras\courses\PP\1920\remote07\simple\simple
this is a char: g (ascii: 103 (HEX:67))
this is a char: h (ascii: 104 (HEX:68))
-----
Process exited after 0.03334 seconds with return value 0
Press any key to continue . . .
```

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 */`
- `'\0' == (char) 0`

Ανάγνωση και εκτύπωση αλφαριθμητικού

- `char str[N_MAX];`
- `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.

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);

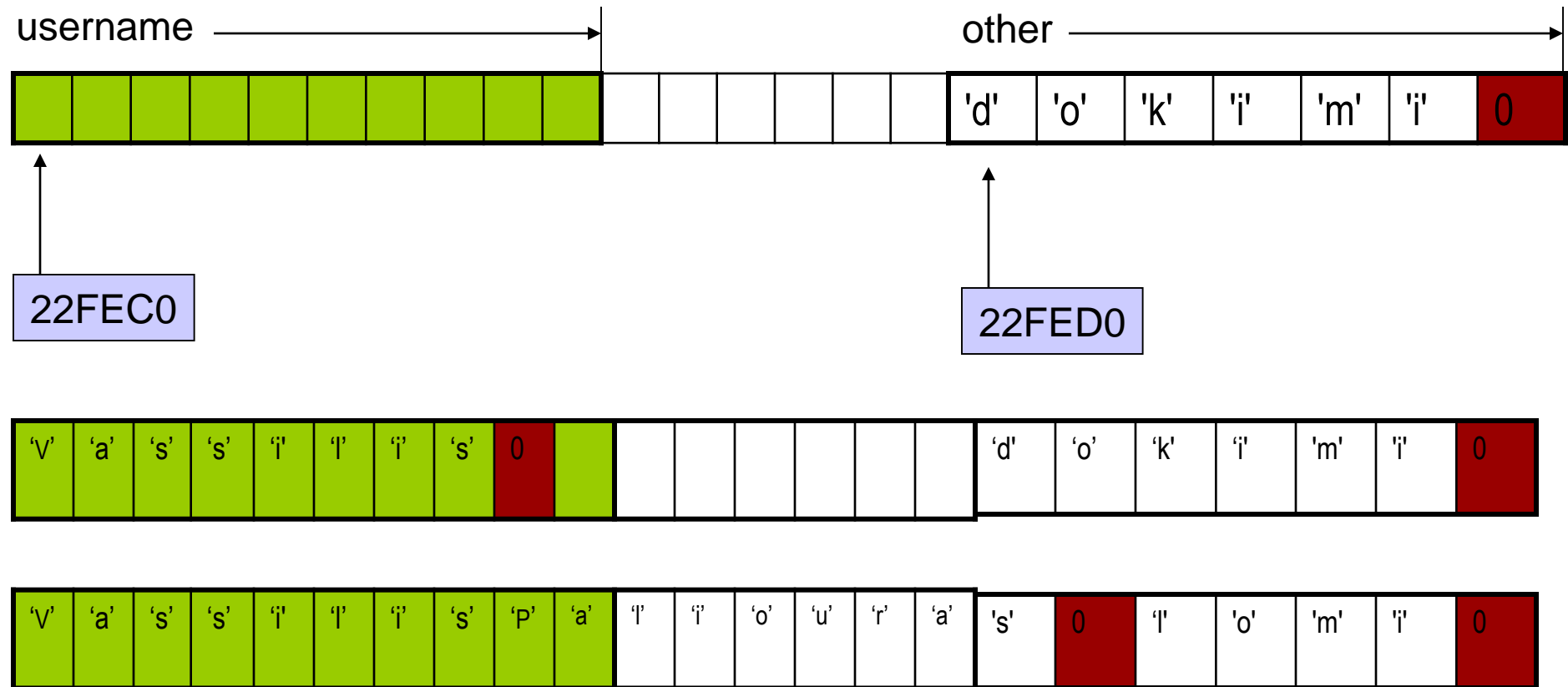
    printf("Hello, %s\n", username);
    printf("Your name is %d letters long stored at %X\n", strlen(username), username);

    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

```
#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;
}
```


Απαρίθμηση με char

```
#include <stdio.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;  
}
```

Strings ως ορίσματα συναρτήσεων

```
#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");
}
```

```
#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

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

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

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

```
    char * ch_ptr ;
```

Δήλωση δείκτη σε χαρακτήρα

```
    ch_ptr = &ch ;
```

```
    ch2 = * ch_ptr ;
```

* ⇒ Περιεχόμενα της θέσης στην οποία δείχνει ο δείκτης

```
    printf ("%c", ch2);
```

```
    return ;
```

```
}
```

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

```
int main(int argc, char *argv[])  
{
```

```
    char ch, ch2;  
    char *ch_ptr ;
```

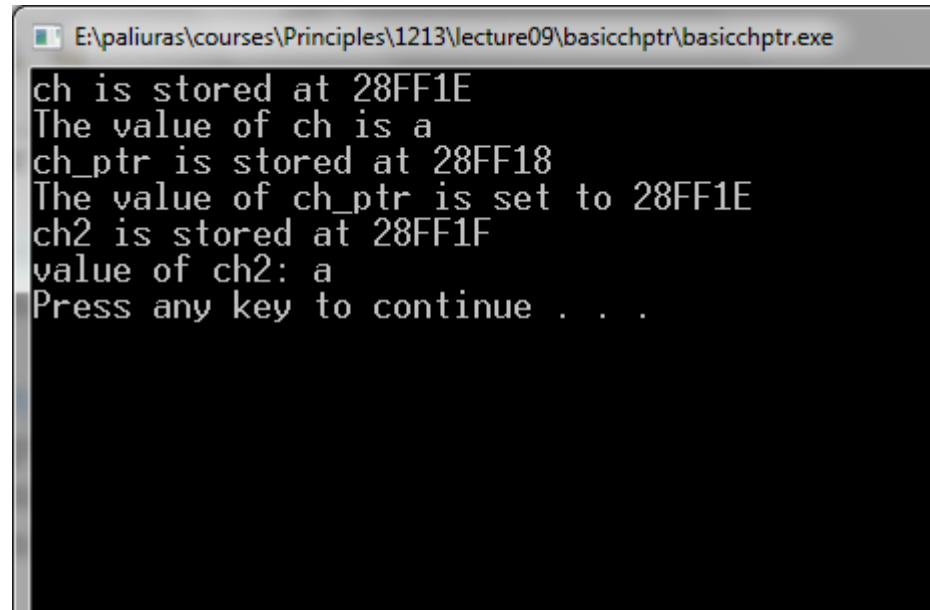
```
    ch = 'a';  
    printf( "ch is stored at %X\n", &ch);  
    printf( "The value of ch is %c\n", ch);
```

```
    /* ch_ptr is set to point to ch */  
    ch_ptr = &ch ;  
    printf( "ch_ptr is stored at %X\n", &ch_ptr);  
    printf( "The value of ch_ptr is set to %X\n", ch_ptr);
```

```
    /* contents of ch_ptr (i.e., value of ch)  
       are copied to ch2 */  
    ch2 = *ch_ptr ;  
    printf ("ch2 is stored at %X\n", &ch2);  
    printf ("value of ch2: %c\n", ch2);
```

```
    return 0;
```

```
}
```



```
E:\paliuras\courses\Principles\1213\lecture09\basicchptr\basicchptr.exe  
ch is stored at 28FF1E  
The value of ch is a  
ch_ptr is stored at 28FF18  
The value of ch_ptr is set to 28FF1E  
ch2 is stored at 28FF1F  
value of ch2: a  
Press any key to continue . . .
```

Χάρτης μνήμης

Διεύθυνση	Περιεχόμενα μνήμης σε hex	Όνομα μεταβλητής
28FF18	1E	ch_ptr
28FF19	FF	
28FF1A	28	
28FF1B	00	
28FF1C		
28FF1D		
28FF1E	61	ch
28FF1F	61	ch2

Πίνακες και δείκτες

- `int arr[10], n ;`
- `*(arr + n) ⇔ arr[n]` `/* δεδομένα
(ακέρατος) */`
- `arr + n ⇔ &arr[n]` `/* διεύθυνση
δεδομένων */`
- `...αλλά και n[arr] ⇔ *(n+arr)` `/* (!!!) */`
- Μπορούμε να χρησιμοποιήσουμε δείκτες για να περάσουμε ως όρισμα σε συνάρτηση πίνακες
 - ακριβέστερα: σε ποια διεύθυνση μνήμης βρίσκεται το πρώτο στοιχείο του πίνακα.


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

```
int main(void) {
    int a[N];
    int i;

    for (i=0; i<N; i++) { a[i] = i*i ;}

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

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

    for (i=0; i<N; i++) {printf("%2d %p %p\n",*(a+i),a+i, &a[i]);}
    printf("\n");

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

    return 0;
}
```

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

```
void myprint(int , int *);
```

```
int main(void) {  
    int x[5] = {1, 2, 3, 4, 5};  
    int y[3] = {5, 10, 15};  
    myprint(5, x);  
    myprint(3, y);  
    return 0;  
}
```

```
void myprint(int len, int * data_ptr) {  
    int i;  
  
    for (i=0; i<len; i++) {  
        printf("%d ", *(data_ptr+i));  
    }  
  
    printf("\n");  
    return ;  
}
```

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 *) ;`
- ...

Δήλωση και ανάθεση τιμής σε αλφαριθμητικό

```
char *strcpy(char *, const char*);
```

```
main ( ) {  
    char name[10];  
    name = "katerina";  
    printf ("%s", name);  
  
    scanf ("%s", name);  
    printf ("%s", name);  
}
```

Λάθος
(στη C)

```
#include <string.h>  
main ( ) {  
    char name[10];  
    strcpy(name, "katerina");  
    printf ("%s", name);  
  
    scanf ("%s", name);  
    printf ("%s", name);  
}
```

Σωστός τρόπος:

Σύγκριση αλφαριθμητικών

- Συνάρτηση

```
int strcmp (const char *, const char *)
```

- για σύγκριση αλφαριθμητικών:

- Επιστρέφει 0 αν είναι ίδια
- -1 αν το πρώτο όρισμα προηγείται του δεύτερου
- +1 αν έπεται.

- **Προσοχή!!!** αν θέλω να συγκρίνω δύο αλφαριθμητικά δεν χρησιμοποιώ τον τελεστή ==

- Αυτός συγκρίνει θέσεις όχι περιεχόμενα!!!

Παράδειγμα σύγκρισης αλφαριθμητικών

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

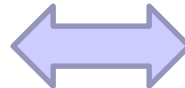
```
int main(int argc, char *argv[])
{
    char name[100] = "Katerina";
    char user[100] = "";

    printf("Enter your name:");

    while (strcmp(user,name)!=0)
        scanf("%s",user);

    printf("Hello %s\n", user);

    return 0;
}
```



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

```
int main(int argc, char *argv[])
{
    char name[100] = "Katerina";
    char user[100] = "";

    printf("Enter your name:");

    while (strcmp(user,name))
        scanf("%s",user);

    printf("Hello %s\n", user);

    return 0;
}
```

Άλλο εννοεί εδώ...

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

```
int main(int argc, char *argv[])  
{
```

```
    char name[100] = "Katerina";  
    char user[100] = "";
```

```
    printf("Enter your name:");  
    while (user != "Katerina") {  
        scanf("%s",user);  
    }
```

```
    printf("Hello %s\n", user);
```

```
    return 0;
```

```
}
```

- **Δεν κάνει** αυτό που χρειάζεται!!!
- Τι κάνει;
 - Συγκρίνει τη θέση που αρχίζει ο user με τη θέση στην οποία είναι αποθηκευμένο το "Katerina".

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(acaunt);
}
```

```
void readstring(char *s ) {
    printf ("alparithmitiko? ");
    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);`

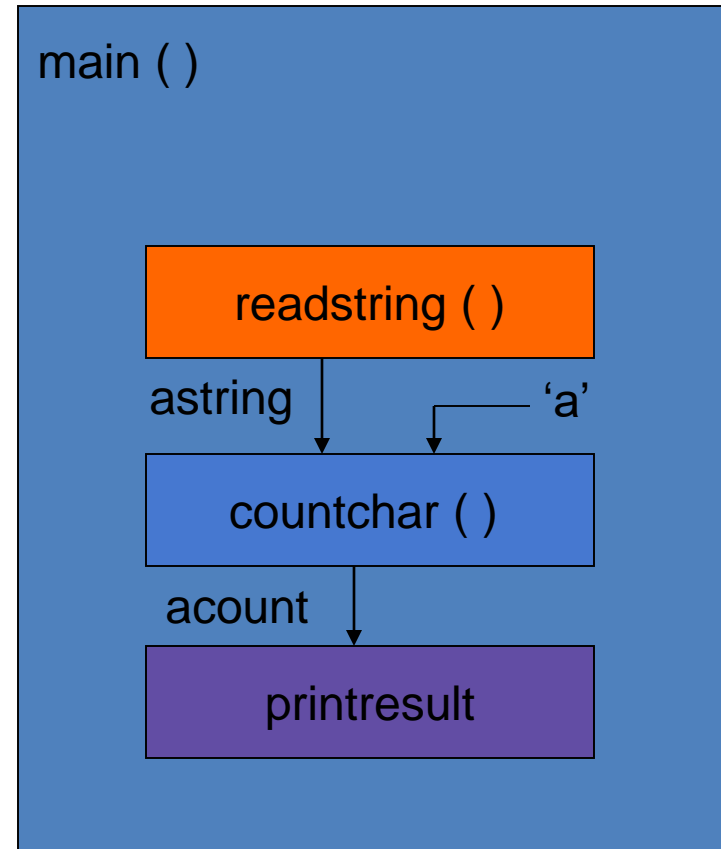
Η `myprint ()` καλείται με διαφορετικό όρισμα (διαφορετική τιμή) στις δύο περιπτώσεις!

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;  
    while ( s[i] )  
        count += (s[i++] == c);  
    return count ;  
}
```

postfix notation
↓

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;  
    while ( s[i] )  
        count += (s[i++] == c);  
    return count ;  
}
```

postfix notation
↓

```
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 swar(int *a, int *b);**
- κλήση: **swar(&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 ;
}
```



```
Paliuras@MOB ~/nlect9
$ ./a
value1: 5 value2: 3
value1: 3 value2: 5
```

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;

Διάρκεια μεταβλητής (2)

```
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);
}
```

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 εμφάνισης 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);
```

```
}
```