

Η γλώσσα C

Διαχείριση Συμβολοσειρών



MYY502

Συμβολοσειρές (strings)

❖ Θυμόμαστε: τι είναι οι συμβολοσειρές;

- Πίνακας χαρακτήρων (με τη σύμβαση ότι στο τέλος έχει '\0').
 - ❖ `char str[10];` /* Άρα 9 χαρακτήρες + το \0 */
- Με εύκολη αρχικοποίηση:
 - ❖ `char str[10] = "This is";`
- Και με διαχείριση που είναι πιο εύκολη από τη διαχείριση πινάκων, όπως θα δούμε.
 - ❖ Όμως μην ξεχνάμε ότι είναι πίνακες (άρα ισχύουν ότι και για αυτούς).

❖ Τι είναι τα παρακάτω:

`char b[10] = "this";`

Το b και το "this" είναι strings.

`char *c;`

Δεν είναι string! Είναι ένας ptr σε χαρακτήρα.

`char *d = "that";`

Το "that" είναι string (αποθηκευμένο κάπου στη μνήμη) και ο d είναι pointer στον πρώτο χαρακτήρα του d. Είναι καταχρηστικό να λέμε ότι και το d είναι string.

```
int main() {  
    char q[] = "First";  
    char r[16] = { 'S', 'e', 'c', 'o', 'n', 'd', '\0' };  
    char s[16] = "Third";    /* Γίνεται αντιγραφή χαρακτήρων. */  
    char t[16];  
    char *u = "Fifth";       /* ΔΕΝ γίνεται κάποια αντιγραφή! */  
    char *w;  
  
    t = "Fourth";    /* Δεν επιτρέπεται! Πρέπει t[0]='F', t[1]='o' ... */  
    w = "Sixth";     /* Επιτρέπεται, μιας και είναι pointer */  
    return 0;  
}
```

❖ `int puts(char *s);`

- Η `puts` τυπώνει το string `s` και ένα χαρακτήρα νέας γραμμής (`'\n'`) στην οθόνη.
- Επιστρέφει EOF αν συμβεί κάποιο λάθος, διαφορετικά επιστρέφει μη αρνητική τιμή.

❖ `int fputs(char *s, FILE *fp);`

- Η `fputs` τυπώνει το string `s` στο αρχείο `fp` (χωρίς επιπλέον `\n`).
- Για τύπωμα στην οθόνη, περάστε ως αρχείο το **`stdout`**.
- Επιστρέφει EOF αν συμβεί κάποιο λάθος, διαφορετικά επιστρέφει μη αρνητική τιμή.
- Π.χ. `fputs("hi", stdout);`

❖ `char *gets(char *s);`

- Η `gets` διαβάζει την επόμενη γραμμή εισόδου και την αποθηκεύει στο string `s`.
- **Αντικαθιστά τον τερματικό χαρακτήρα νέας γραμμής με `'\0'`.**
- Επιστρέφει `s`, ή `NULL` αν συναντηθεί το τέλος του αρχείου (EOF) ή αν συμβεί κάποιο λάθος.
- **Δεν είναι ασφαλής συνάρτηση** (τι γίνεται αν το `s[]` δεν φτάνει?).

❖ `char *fgets(char *s, int num, FILE *fp);`

- Η `gets` διαβάζει **μέχρι num-1 χαρακτήρες** από το αρχείο `fp` (το πληκτρολόγιο είναι το `stdin`) και την αποθηκεύει στο string `s`.
- **Αν πληκτρολογήθηκε το newline, τότε γράφεται ΚΑΙ ο χαρακτήρας `\n` μέσα στο `s` και αμέσως μετά γράφεται και το `'\0'`.**
- Επιστρέφει `s`, ή `NULL` αν συναντηθεί το τέλος του αρχείου (EOF) ή αν συμβεί κάποιο λάθος.
- **Να την προτιμάτε έναντι της `gets()`.**
- Π.χ. `fgets(str, 9, stdin); /* έως 8 χαρακτήρες από πληκτρολόγιο */`

❖ Εκτύπωση ή διάβασμα με το "%s" στο format

❖ printf:

- "%s" : τυπώνει όλο το string
- "%Ns" τυπώνει τουλάχιστον N χαρακτήρες (με κενά αν χρειαστεί)
- "%.Ms" τυπώνει το πολύ M χαρακτήρες
- "%N.Ms" τυπώνει τουλάχιστον N και το πολύ M χαρακτήρες
- "%-Ns" τυπώνει τουλάχιστον N χαρακτήρες, με αριστερή στοίχιση
- Π.χ. printf("%-3.5s", str);

❖ scanf:

- "%s": διαβάζει μία συμβολοσειρά
- **ΠΡΟΣΟΧΗ:** η συμβολοσειρά τελειώνει όταν συναντηθεί χαρακτήρας κενού (*space, tab, newline*)

```
/* gets example */  
#include <stdio.h>  
int main() {  
    char string [256];  
    printf ("Insert your full address: ");  
    gets(string);  
    printf("Your address is: %s\n", string);  
    return 0;  
}
```

```
/* puts example */  
#include <stdio.h>  
int main () {  
    char string [] = "Hello world!";  
    puts(string);  
}
```

Παραδείγματα

```
#include <stdio.h>
int main() {
    char test1[5], test2[5];
    scanf("%s", test1);
    printf("test1=%s\n", test1);
    gets(test2);
    printf("test2=%s\n", test2);
    return 0;
}
```

Εκτέλεση:

\$./a.out

1313

test1=1313

test2=

\$...

Το test2 θα είναι ίσο με "end of line" από την προηγούμενη είσοδο διότι η `scanf( )` **ολοκληρώνει το διάβασμά της μόλις συναντήσει τον πρώτο κενό χαρακτήρα** (space, tab, newline) – αλλά δεν τον «καταναλώνει»! Έτσι η `gets( )` βρίσκει το newline και επιστρέφει αμέσως...

Παραδείγματα

```
#include <stdio.h>
int main() {
    char test1[5], test2[5];
    scanf("%s", test1);
    printf("test1=%s\n", test1);
    scanf("%s", test2);
    printf("test2=%s\n", test2);
    return 0;
}
```

Εκτέλεση:

\$./a.out

1313

test1=1313

1233

test2=1233

\$...

Χρησιμοποιείται η `scanf()` αντί για την `gets()`. Η `scanf("%s", )` **στην αρχή του διαβάσματος αγνοεί τα κενά** (space, tab, newline) και άρα την αλλαγή γραμμής. Στη συνέχεια διαβάζει από το πρώτο μη-blank και σταματά να διαβάζει στο αμέσως επόμενο blank.

```
#include <stdio.h>
int main() {
    char lula[]="lula";
    char *ptr = lula;
    puts(lula); puts(" > ");
    puts(ptr + 2);
    printf("%s > %s\n", lula, ptr+2);
    return 0;
}
```

Εκτέλεση:

\$./a.out

lula

>

la

lula > la

\$...

Η puts προσθέτει το \n

- ❖ «Χειροποίητες» puts που δεν εκτυπώνουν το EOL.
 - Η putchar(ch) τυπώνει στην οθόνη τον χαρακτήρα ch
 - Είναι ισοδύναμη με το printf("%c", ch);

```
void myputs_array(char *string) {  
    int i = 0;  
    while (string[i] != '\0') {  
        putchar(string[i]);  
        i++;  
    }  
}
```

```
void myputs_pointer(char *string) {  
    while (*string != '\0') {  
        putchar(*string);  
        string++;  
    }  
}
```

```
/* Το '\0' έχει ASCII κωδικό μηδέν (0) ! */  
void myputs_oneliner(char *string) {  
    while (*string) putchar(*(string++));  
}
```

❖ `int sprintf(char *s, char *format, ...)`

- Η συνάρτηση αυτή λειτουργεί ακριβώς όπως η `printf()` με τη διαφορά ότι **δεν τυπώνει στην οθόνη αλλά γράφει στο string `s` στο οποίο τοποθετείται επιπλέον και ο χαρακτήρας `'\0'`.**
- Επιστρέφεται ο αριθμός των χαρακτήρων που γράφτηκαν στο `s` πλην του χαρακτήρα `'\0'`.

❖ `int sscanf(char *s, char *format, ...)`

- Η συνάρτηση αυτή λειτουργεί ακριβώς όπως η `scanf()` με τη διαφορά ότι **η είσοδος των δεδομένων προέρχεται από το string `s` και όχι από το πληκτρολόγιο.**
- Επιστρέφει είτε τον αριθμό των αντικειμένων που ενημερώθηκαν (αν όλα πάνε καλά), είτε EOF (σε περίπτωση λάθους ή τέλους της συμβολοσειράς `s`).

Παράδειγμα

```
/* sprintf example */
#include <stdio.h>
int main () {
    char buffer[50];
    int n, a=5, b=3;

    n=sprintf(buffer, "%d plus %d is %d", a, b, a+b);
    printf("[%s] is a %d char string\n", buffer, n);
    return 0;
}
```

```
$ ./a.out
```

```
[5 plus 3 is 8] is a 13 char string
```

Παράδειγμα

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

```
int main() {
```

```
    int k, m;
```

```
    float f;
```

```
    char *x="2 minutes to 12.0";
```

```
    char y[20], z[20], w[80];
```

```
    sscanf(x, "%d%s%s%f", &m, y, z, &f) ;
```

```
    printf("%d\n%s\n%s\n%f\n", m, y, z, f) ;
```

```
    k = sprintf(w, "%d %s %d %s ", m, z, (int) f, y) ;
```

```
    printf("\nNew order: %s\n", w);
```

```
    printf("with %d characters (including spaces)\n", k);
```

```
    return 0;
```

```
}
```

\$/a.out

2

minutes

to

12.000000

New order: 2 to 12 minutes

with 15 characters (including spaces)

Επαναληπτική κλήση της sscanf

```
#include <stdio.h>

int main() {
    int k;
    char x[30]="2 minutes to 12.0";
    char *p;
    char y[30];

    p = x;
    while (sscanf(p, "%s", y) > 0) {
        k = printf("%s\n", y); /* # chars in y, +1 */
        p = p + k;             /* skip k chars */
    }
    return 0;
}
```

Επαναληπτική μέτρηση λέξεων

```
#include <stdio.h>

int main() {
    int k, count;
    char x[30], y[30], *p;

    while (1) {
        count = 0;
        if (fgets(x, 30, stdin) == NULL) /* Ctrl-D */
            break;
        p = x;
        while (sscanf(p, "%s", y) > 0) {
            k = printf("%s\n", y);
            p = p + k;
            count++;
        }
        printf("Total number of words: %d\n", count);
    }
    return 0;
}
```


Επαναληπτική μέτρηση λέξεων (συντομότερη)

```
#include <stdio.h>

int main() {
    int count;
    char x[90], y[90], *p;

    while (fgets(x, 90, stdin) != NULL) {
        for (count = 0, p = x; sscanf(p, "%s", y) > 0; count++) {
            p = p + printf("%s\n", y);
        }
        printf("Total number of words: %d\n", count);
    }
    return 0;
}
```

- ❖ Μια συμβολοσειρά (string) είναι ένας πίνακας χαρακτήρων στον οποίο τοποθετείται τελευταίος ο χαρακτήρας '\0', ως ένδειξη του τέλους της συμβολοσειράς
- ❖ Μπορούμε να διαχειριστούμε ένα string με δύο τρόπους
 - Ως έναν πίνακα, το οποίο συνεπάγεται σχετική δυσκολία

```
char line[8];  
line[0] = 'H'; line[1] = 'e'; line[2] = 'l'; line[3] =  
'l'; line[4] = 'o'; line[5] = '\0';
```
 - Μέσω της χρήσης ειδικών συναρτήσεων που παρέχει η C μέσω του αρχείου **<string.h>**

```
strcpy(line, "Hello");
```

❖ `char *strcpy(s, t);`

➤ Αντιγράφει το string `t` στο `s`, μαζί με τον χαρακτήρα `'\0'` και επιστρέφει το `s`

➤ Παράδειγμα

```
char s[6];  
strcpy(s, "hello");
```

➤ Τι γίνεται αν στο `s` δεν χωράει το `t`;

❖ `char *strncpy(s, t, n);`

➤ Αντιγράφει το πολύ `n` χαρακτήρες από το `t` στο `s`, το `t` μπορεί να έχει λιγότερους. Επιστρέφει το `s`

❖ `char *strcat(s, t);`

➤ Προσθέτει στο τέλος του `s` το string `t`. Επιστρέφει το `s`.

❖ `char *strncat(s, t, n);`

➤ Προσθέτει στο τέλος του `s` το πολύ `n` χαρακτήρες του `t`, και τοποθετεί επίσης και τον χαρακτήρα `'\0'`. Επιστρέφει το `s`.

❖ `int strcmp(s, t);`

- Συγκρίνει λεξικογραφικά τα δύο strings.
 - ✧ (βασικό κριτήριο) περιεχόμενο
 - ✧ (δευτερεύον κριτήριο) μήκος
- Επιστρέφει:
 - ✧ $\text{Av } (s == t) \rightarrow 0$
 - ✧ $\text{Av } (s > t) \rightarrow \text{θετικό}$
 - ✧ $\text{Av } (s < t) \rightarrow \text{αρνητικό}$

❖ `int strncmp(s, t, n);`

- Όπως και παραπάνω αλλά συγκρίνει λεξικογραφικά το πολύ n χαρακτήρες

❖ `char *strstr(s, t);`

- Επιστρέφει ένα δείκτη στην πρώτη εμφάνιση στο s του t , (διαφορετικά) επιστρέφει NULL αν το t δεν περιέχεται στο s .

❖ `int strlen(s);`

- Επιστρέφει το μήκος της συμβολοσειράς s (χωρίς το $\backslash 0$).

❖ Έστω:

- `char a[30] = "Kalimera, ";`
- `char b[20] = "Kalo mathima!";`

❖ Τότε:

- `strcpy(a,b)`
 - ✧ `printf("%s", a);` $\rightarrow$ *Kalo mathima!*
- `strncpy(a,b,4)`
 - ✧ `printf("%s", a);` $\rightarrow$ *Kalomera,*
- `strcat(a,b)`
 - ✧ `printf("%s", a);` $\rightarrow$ *Kalimera, Kalo mathima!*

❖ Έστω:

- `int ret; char *p;`
- `char a[30] = "Kalimera, ";`
- `char b[20] = "Kalo mathima!";`

❖ Τότε:

- `ret = strcmp(a,b);`
 - ❖ `printf("%d", ret);` → κάποια αρνητική τιμή
- `p = strstr(a,"im");`
 - ❖ `printf("%s", p);` → `imera,`
- `printf("%d", strlen(a) );` → `10`

❖ Τι θα τυπώσει το: `printf("%d, %d", sizeof(a), strlen(a));`

- `30, 10`

- ❖ `char *strtok(char *s, char *t);`
 - Ψάχνει στο `s` για κομμάτια (tokens) που διαχωρίζονται με τους χαρακτήρες που περιγράφονται στο `t`.
 - Κάθε διαφορετική κλήση της `strtok` επιστρέφει και ένα καινούργιο token (κανονικό string με χαρακτήρα τερματισμού).
 - Χρήση:
 - ✧ 1^ο token: καλώ `tok = strtok(s, t);`
 - ✧ Επόμενα (συνήθως σε loop): καλώ `tok = strtok(NULL, t);`
 - Επιστρέφει `NULL` αν δεν υπάρχουν άλλα tokens στο `s`.

Προσοχή!

- ***Η συνάρτηση τροποποιεί το πρώτο όρισμα της***
- ***Δεν μπορεί να χρησιμοποιηθεί σε σταθερά strings***

Παράδειγμα strtok

```
char email[] = "zas@cse.uoi.gr";  
char token[] = "@";  
char *s;  
  
s = strtok(email, token);  
s = strtok(NULL, token);  
...
```



Παράδειγμα strtok (1/2)

- ❖ Να γράψετε πρόγραμμα το οποίο λαμβάνει διευθύνσεις ηλεκτρονικού ταχυδρομείου και επιστρέφει τα πεδία από τα οποία αποτελούνται
- ❖ Σχετικό παράδειγμα εκτέλεσης:

```
$ myprog
```

```
type email address: zas@cse.uoi.gr
```

```
fields of email address: zas, cse, uoi, gr
```

Παράδειγμα strtok (2/2)

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

int main() {
    char email[80];
    char token[] = "@.";
    char *s;

    printf("type email address:");
    scanf("%s", email);
    printf("fields of email address:");
    s = strtok(email, token);
    if (s != NULL) {
        printf("%s", s);
    }
    while ((s = strtok(NULL, token)) != NULL) {
        printf(", %s", s);
    }
    return 0;
}
```



Συναρτήσεις Ελέγχου - Μετατροπής

❖ `#include <ctype.h>`

❖ Συναρτήσεις ελέγχου

- `int isalnum(int c);` true για γράμμα ή ψηφίο
- `int isalpha(int c);` true για γράμμα
- `int isdigit(int c);` true για ψηφίο
- `int isspace(int c);` true για κενό, tab, \n, ...
- `int islower(int c);` true για γράμμα μικρό
- `int isupper(int c);` true για γράμμα κεφαλαίο

❖ Συναρτήσεις μετατροπής

- `int tolower(int c);` μετατροπή κεφαλαίου σε μικρό
- `int toupper(int c);` μετατροπή μικρού σε κεφαλαίο

❖ #include <stdlib.h>

- `int atoi(char *s);` μετατροπή string σε ακέραιο
 - ✧ Το string s πρέπει να ξεκινά με κενό ή κάποιον αριθμό
 - ✧ Η συνάρτηση σταματά να διαβάζει από το string μόλις βρει κάποιον μη-αριθμητικό χαρακτήρα
 - ✧ Αν η μετατροπή δεν μπορεί να συμβεί, επιστρέφει 0
- `long atol(char *s);` μετατροπή string σε long
- `double atof(char *s);` μετατροπή string σε double

❖ Πώς μετατρέπω αριθμούς σε strings;

- *Homework!*

- ❖ `int i;`
`i = atoi( "512" );`
`i = atoi( "512.035" );`
`i = atoi( " 512.035" );`
`i = atoi( " 512+34" );`
`i = atoi( " 512 bottles of beer on the wall" );`
- ❖ `int i = atoi( " does not work: 512" ); // → i = 0`
- ❖ `long l = atol( "1024.0001" );`
- ❖ `double x = atof( "42.0is_the_answer" );`

❖ Βασικές συναρτήσεις εισόδου – εξόδου

- `int printf(char *format, ...);`
- `int scanf(char *format, ...);`

❖ Ειδικοί χαρακτήρες στο format τους

➤ Ακέραιοι αριθμοί

- ✧ `%d` στο δεκαδικό σύστημα
- ✧ `%u` χωρίς πρόσημο στο δεκαδικό σύστημα
- ✧ `%o` χωρίς πρόσημο στο οκταδικό σύστημα
- ✧ `%x` χωρίς πρόσημο στο δεκαεξαδικό σύστημα

➤ Αριθμοί κινητής υποδιαστολής

- ✧ `%f` σε μορφή: `[-]ddd.dddddd`
- ✧ `%e` σε μορφή: `[-]ddd.dddddd e[+/-]ddd`
- ✧ `%g` σε μορφή `%f` ή `%e`

❖ Ειδικοί χαρακτήρες στο format τους

➤ Άλλοι τύποι

- ✧ %c χαρακτήρας
- ✧ %s συμβολοσειρά (string)
- ✧ %p δείκτης

❖ Παραλλαγές στο format

➤ Μέγεθος αριθμών

- ✧ %h αριθμοί short, π.χ. %hd, %hx
- ✧ %l αριθμοί long ή double, π.χ. %ld, %lf
- ✧ %L αριθμοί long double, π.χ. %Lf

❖ Παραλλαγές στο format

➤ Μήκος αποτελέσματος

- ✧ %8d αριθμός σε μήκος 8 χαρακτήρων
- ✧ %20s συμβολοσειρά σε μήκος 20 χαρακτήρων
- ✧ %+8d αριθμός σε μήκος 8 χαρακτήρων, τύπωσε και πρόσημο
- ✧ %08d αριθμός σε μήκος 8 χαρακτήρων, τα πρώτα εξ' αυτών 0
- ✧ %-8d όπως το %8d με στοίχιση αριστερά

printf demo (1/2)

```
#include <stdio.h>
int main() {
    int    i;        // Number to print.
    double f;        // Number to print.

    printf("Enter an integer (use either + or -): ");
    scanf ("%d", &i);
    printf("This is the integer.....%d\n", i);
    printf("This is the integer in octal.....%o\n", i);
    printf("Octal with leading zero.....%#o\n", i);
    printf("This is the integer in hex.....%x| or |%X|\n", i, i);
    printf("Hex with leading 0x.....%#x| or |%#X|\n", i, i);
    printf("Forcing a plus or minus sign.....%+d \n", i);
    printf("Include space before + numbers.....% d \n", i);
    printf("Field width of 3.....%3d \n", i);
    printf("Field width of 5.....%5d \n", i);
    printf("Field width of 7.....%7d \n", i);
    printf("Same as above with left justification. %-7d\n", i);
    printf("Field width of 7 with zero fill.....%07d \n", i);
    printf("At least 3 digits.....%.3d \n", i);
    printf("At least 5 digits.....%.5d \n", i);
    printf("Field width of 10, at least 7 digits..%10.7d\n", i);
```

printf demo (2/2)

```
printf("\nEnter a floating point number: ");
scanf ("%lf", &f);

printf("This is the number.....|%f|\n", f);
printf("Forcing a plus or minus sign.....| %+f|\n", f);
printf("Field width of 20.....| %20f|\n", f);
printf("0 decimal places.....| %.0f|\n", f);
printf("0 decimal places forcing decimal.....| %#.0f|\n", f);
printf("3 decimal places.....| %.3f|\n", f);
printf("20 decimal places.....| %.20f|\n", f);
printf("Field width of 20, 3 decimal places...| %20.3f|\n", f);
printf("\nHere is the number in e format:\n");
printf("3 decimal places.....| %.3e|\n", f);
printf("5 decimal places & big 'e'.....| %.5E|\n", f);

printf("\nHere is the number in g format:\n");
printf("No special requests.....| %g|\n", f);
printf("Maximum of 1 significant figure.....| %.1g|\n", f);
printf("Maximum of 4 significant figures.....| %.4g|\n", f);

return 0;
}
```

Τα παρακάτω περιγράφουν πλήρως την μερικές φορές «περίεργη» συμπεριφορά της scanf():

❖ scanf("%c" ...)

- Διαβάζει αμέσως όποιον χαρακτήρα βρει (ακόμα και κενό) και επιστρέφει

❖ scanf("%<οτιδήποτε άλλο>" ...)

- Αγνοεί τα αρχικά κενά (space, tab, newline) και αρχίζει και διαβάζει μόλις συναντήσει μη-κενό χαρακτήρα
- Τελειώνει και επιστρέφει μόλις συναντήσει κενό χαρακτήρα. Όμως, δεν τον καταναλώνει! – θα τον βρει η επόμενη scanf.

Τα παρακάτω περιγράφουν πλήρως την μερικές φορές «περίεργη» συμπεριφορά της scanf():

❖ scanf("Give number: %d" ...)

- ΔΕΝ ΔΟΥΛΕΥΕΙ διότι: οτιδήποτε είναι εκτός των “%” **θα πρέπει να το πληκτρολογήσει επακριβώς ο χρήστης – δεν τυπώνονται στην οθόνη!**

❖ scanf("%c", ...)

- Διαβάζει αμέσως όποιον χαρακτήρα βρει (ακόμα και κενό) και επιστρέφει – Πώς μπορώ να διαβάσω τον επόμενο μη-κενό χαρακτήρα όπως όλα τα αλλά “%”;
- Απάντηση 1: βάζω 1 κενό πριν το %: scanf(" %c", ...)
- Απάντηση 2: πιο «στάνταρ»: scanf("%1s", ...)

❖ Διαχείριση χαρακτήρων/συμβολοσειρών

- `int putchar(int c);`
- `int getchar();`
- `int puts(char *s);` `/* also: fputs() */`
- `char *gets(char *s);` `/* unsafe, prefer: fgets() */`

❖ Διαχείριση συμβολοσειρών <string.h>

- `size_t strlen(char *s):`
Μέτρηση αριθμού χαρακτήρων της συμβολοσειράς s
- `char *strcpy(char *s1, const char *s2):`
Αντιγραφή της συμβολοσειράς s2 στην s1
- `char *strcat(char *s1, const char *s2):`
Προσθήκη της συμβολοσειράς s2 στο τέλος της s1
- `int strcmp(char *s1, const char *s2):`
Σύγκριση των συμβολοσειρών s1 και s2

- ❖ Μετατροπή συμβολοσειρών `<stdlib.h>`
 - `int atoi(char *s):`
Μετατροπή της συμβολοσειράς `s` σε `int`.
 - `long int atol(char *s):`
Μετατροπή της συμβολοσειράς `s` σε `long int`.
 - `double atof(char *s):`
Μετατροπή της συμβολοσειράς `s` σε `double`.

Προγραμματισμός σε C

*Περίπτωση διαχείρισης συμβολοσειρών:
Ορίσματα στη main()*



ΜΥΥ 502

Ορίσματα στην main()

```
$ ls
```

```
$ ls -l; ls -al; gcc myfile.c -lm
```

- ❖ Γενικά σε ένα πρόγραμμα μπορούμε να δώσουμε ως είσοδο δεδομένα/ορίσματα/επιλογές τη στιγμή που ξεκινάει η εκτέλεση του, από τη γραμμή εντολών
- ❖ Πώς μπορούμε να γνωρίζουμε τα δεδομένα/ορίσματα που δίνει ο χρήστης;
 - *Απάντηση:*
Παράμετροι στην main()! (την οποία μέχρι τώρα την ορίζαμε χωρίς παραμέτρους)
- ❖ Τα δεδομένα τα δέχεται η main() ως strings

Ορίσματα στην main()

- ❖ Μέχρι τώρα:

```
int main() { ... }
```

- ❖ Γενικά όμως, ο προγραμματιστής μπορεί να γράψει:

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

ή ισοδύναμα:

```
int main(int argc, char **argv) { ... }
```

- ❖ Το `argv` είναι ένας πίνακας δεικτών σε συμβολοσειρές (strings)
- ❖ Το `argc` είναι το πλήθος των στοιχείων του πίνακα
- ❖ Πάντα το στοιχείο 0 είναι το όνομα του προγράμματος

Παράδειγμα ορισμάτων στην main()

```
$ a.out hi there
```

- ❖ `argc = 3`
- ❖ `argv[0]`: όνομα προγράμματος, "a.out"
- ❖ `argv[1]`: πρώτο όρισμα προγράμματος "hi"
- ❖ `argv[2]`: δεύτερο όρισμα προγράμματος "there"

Παράδειγμα (εκτύπωση ορισμάτων)

```
$ a.out hello world  
argc = 3  
Program name: a.out  
Arguments: hello, world,
```

```
#include <stdio.h>  
int main(int argc, char *argv[]) {  
    int i;  
    printf("argc = %d\n", argc);  
    printf("Program name: %s\n", argv[0]);  
    printf("Arguments: ");  
    for (i = 1; i < argc; i++)  
        printf("%s, ", argv[i]);  
    printf("\n");  
    return 0;  
}
```

Επιπλέον παράδειγμα (πρόσθεση ορισμάτων)

\$ myadd 5 10 15

Program name: myadd

Result: 30

```
#include <stdio.h>
#include <stdlib.h>    /* because of atoi() */
int main(int argc, char *argv[]) {
    int i, sum = 0;
    printf("Program name: %s\n", argv[0]);
    if (argc < 2) exit(1);    /* Nothing to add */
    for (i = 1; i < argc; i++) {
        sum = sum + atoi(argv[i]);
    }
    printf("Result: %d\n", sum);
    return 0;
}
```