

Δείκτες και Μονοδιάστατοι Πίνακες [1/3]

Μια δήλωση της μορφής:

```
int a[4];
```

προκαλεί δέσμευση τεσσάρων συνεχόμενων θέσεων μνήμης για την αποθήκευση των στοιχείων του μονοδιάστατου πίνακα με όνομα *a*.

Οι διευθύνσεις αυτών των στοιχείων θα είναι:

&a[0], &a[1], &a[2] και &a[3].

Το όνομα του πίνακα είναι ένας σταθερός δείκτης που περιέχει τη διεύθυνση του πρώτου στοιχείου του

(*a* == &a[0]).

Αυτό το επαληθεύουμε εύκολα με το ακόλουθο πρόγραμμα:

```
int main()
{
    int i, a[4]={0}, b[4];

    for(i=0; i<4; i++)
    {
        printf("&a[%d] = %u : a[%d] = %d, ", i, &a[i], i, a[i]);
        printf("&b[%d] = %u : b[%d] = %d\n", i, &b[i], i, b[i]);
    }
    printf("\na = %u, b = %u\n", a, b);
    return 0;
}
```

Δείκτες και Μονοδιάστατοι Πίνακες [2/3]

Εξοδος προγράμματος:

```
&a[0] = 2293584 : a[0] = 0, &b[0] = 2293568 : b[0] = 2009095316
&a[1] = 2293588 : a[1] = 0, &b[1] = 2293572 : b[1] = 2008948848
&a[2] = 2293592 : a[2] = 0, &b[2] = 2293576 : b[2] = -1
&a[3] = 2293596 : a[3] = 0, &b[3] = 2293580 : b[3] = 2009055971
```

a = 2293584, b = 2293568

↑
"σκουπίδια"

Προσπέλαση στοιχείων πίνακα

Η προσπέλαση του *i* στοιχείου ενός πίνακα *a* μπορεί να γίνει με δύο τρόπους:

α) με χρήση της αναφοράς *a[i]*, ή β) μέσω αριθμητικής δεικτών: **(a+i)*

Παραδείγματα:

```
a[0] = 12; <==> *a = 12;
```

```
a[1] = 31; <==> *(a+1) = 31;
```

```
a[5] = a[3]; <==> *(a+5) = *(a+3);
```

```
a[i] = a[i-1] + a[i-2]; <==> *(a+i) = *(a+i-1) + *(a+i-2);
```

Δείκτες και Μονοδιάστατοι Πίνακες [3/3]

```
#include <stdio.h>

int main()
{
    int pin[4]={10,20,30,40}, *ptr1, i;
    double A[4]={0.1,0.2,0.3,0.4}, *ptr2;

    ptr1 = pin; /* Εκχώρηση διεύθυνσης pin στον ptr1 */
    ptr2 = A;    /* Εκχώρηση διεύθυνσης A στον ptr2 */

    for(i=0; i<4; i++)
    {
        printf("ptr1+%d: %8u  %d  --  ",i,ptr1+i,*(ptr1+i));
        printf("ptr2+%d: %8u  %.1f\n",i,ptr2+i,*(ptr2+i));
    }

    return 0;
}
```

Εξοδος προγράμματος:

```
ptr1+0: 1245012 10  -- ptr2+0: 1244976 0.1
ptr1+1: 1245016 20  -- ptr2+1: 1244984 0.2
ptr1+2: 1245020 30  -- ptr2+2: 1244992 0.3
ptr1+3: 1245024 40  -- ptr2+3: 1245000 0.4
```

Διαφορά μεταξύ πίνακα και δείκτη

Η δήλωση `int a[100];`

- δημιουργεί έναν δείκτη με όνομα `a`,
- δεσμεύει 100 θέσεις μνήμης τύπου `int`,
- αρχικοποιεί τον δείκτη στη διεύθυνση της πρώτης θέσης μνήμης και
- δεν επιτρέπει την μεταβολή του κατά την εκτέλεση του προγράμματος (σταθερός δείκτης).

Η δήλωση `int *a;` δημιουργεί μια μεταβλητή δείκτη με όνομα `a` αλλά δεν τον αρχικοποιεί ούτε δεσμεύει άλλες θέσεις μνήμης.

Προσοχή!!

Στη C ο έλεγχος των ορίων ενός πίνακα είναι ευθύνη του προγραμματιστή.

Για παράδειγμα, έστω ότι δηλώσαμε έναν πίνακα 100 ακεραίων:

```
int i, a[100];
```

και έστω ότι δεν ήμασταν προσεκτικοί όταν αρχικοποιήσαμε τον πίνακα αυτό μέσα από ένα βρόχο:

```
for(i=1; i<=100; i++) /* αντί του for(i=0; i<100; i++) */
    a[i] = i*i;
```

με αποτέλεσμα κατά την τελευταία επανάληψη, η τιμή του `i` να είναι 100 και η εκχώρηση του **1002** να γίνει στο `a[100]` αντί του `a[99]`. Αυτό επιτρέπεται στις περισσότερες εκδόσεις της C και σχεδόν πάντα έχει καταστροφικά αποτελέσματα γιατί μπορεί να γράψει πάνω σε άλλα δεδομένα του προγράμματός μας!!

Περισσότερα για μεταβίβαση πινάκων σε συναρτήσεις

```
#include <stdio.h>
int main() {
    int array[100];
    ...
    printarray(array);
    return 0;
}

void printarray(int p[]) /* ή void printarray(int *p) */ {
    int i;
    for(i=0; i<100; i++)
        printf("%d\n",p[i]); /* ή printf("%d\n",*(p+i)); */
}
```

ΣΥΜΒΟΛΟΣΕΙΡΕΣ (STRINGS) [1/3]

- Μια συμβολοσειρά είναι μια ακολουθία ενός ή περισσότερων χαρακτήρων και αποτελεί έναν από τους πλέον χρήσιμους και σημαντικούς τύπους δεδομένων της C.
- Όπως γνωρίζουμε ήδη, τα διπλά εισαγωγικά χρησιμοποιούνται για να ορίσουμε μια συμβολοσειρά ακριβώς όπως χρησιμοποιούμε και τα απλά εισαγωγικά για να ορίσουμε έναν χαρακτήρα. Για παράδειγμα, η συμβολοσειρά
 - "A C-program"
- αποτελείται από τους 11 χαρακτήρες 'A',' ','C','-','p','r','o','g','r','a','m' ενώ τα διπλά εισαγωγικά δεν αποτελούν μέρος της συμβολοσειράς.
- Στη C, δεν υπάρχει κάποιος νέος τύπος δεδομένων για τον χειρισμό συμβολοσειρών. Οι συμβολοσειρές αποθηκεύονται σε μονοδιάστατους πίνακες τύπου char. Όμως, προκειμένου να είμαστε σε θέση να υπολογίζουμε το μήκος τους, όπως, για παράδειγμα, κάνει η συνάρτηση strlen(), ή να γράφουμε συναρτήσεις που να τις χειρίζονται, σηματοδοτούμε το τέλος τους με έναν ειδικό χαρακτήρα. Αυτός είναι ο μη εκτυπώσιμος **μηδενικός χαρακτήρας (null character)** '\0' που ακολουθεί τον τελευταίο χαρακτήρα της συμβολοσειράς.
- **Προσοχή:** Ο μηδενικός χαρακτήρας έχει ASCII κωδικό 010 ή 08 ή 000 00002 και είναι εντελώς διαφορετικός από το ψηφίο 0 με ASCII κωδικό 4810 ή 608 ή 110 00002 ('0' ≡ '\60').

ΣΥΜΒΟΛΟΣΕΙΡΕΣ (STRINGS) [2/3]

- Η δήλωση:
`char symb[] = {'A', ' ', 'C', '-', 's', 't', 'r', 'i', 'n', 'g'};`
- αποθηκεύει τον πίνακα χαρακτήρων **symb** σε 10 bytes μνήμης ως εξής:

A		C	-	s	t	r	i	n	g
---	--	---	---	---	---	---	---	---	---

- ενώ η συμβολοσειρά **"A C-string"** αποθηκεύεται σε 11 bytes μνήμης ως :

A		C	-	s	t	r	i	n	g	\0
---	--	---	---	---	---	---	---	---	---	----

- Και στις δύο περιπτώσεις έχουμε αποθηκεύσει την ίδια ακολουθία χαρακτήρων μόνο που στη δεύτερη περίπτωση μπορούμε εύκολα να υπολογίζουμε το μήκος της, μετρώντας το πλήθος των χαρακτήρων μέχρι το '\0', χωρίς να χρειάζεται να "περνάμε" το μήκος της σε συναρτήσεις που χειρίζονται συμβολοσειρές. Τα διπλά εισαγωγικά λένε στον compiler να προσθέσει τον μηδενικό χαρακτήρα στο τέλος.

ΣΥΜΒΟΛΟΣΕΙΡΕΣ (STRINGS) [3/3]

- Γνωρίζουμε από τα προηγούμενα ότι το όνομα ενός πίνακα χαρακτήρων είναι ένας σταθερός δείκτης σε char. Αν θέλουμε δε να χρησιμοποιούμε συναρτήσεις που χειρίζονται συμβολοσειρές θα πρέπει να φροντίσουμε εμείς οι ίδιοι έτσι ώστε ο τελευταίος χαρακτήρας να είναι ο '\0'.
- Για παράδειγμα, θα έπρεπε να δηλώσουμε τη συμβολοσειρά **symb** ως εξής:
`char symb[] = {'A', ' ', 'C', '-', 's', 't', 'r', 'i', 'n', 'g', '\0'};`
- ώστε να μπορούμε να την εμφανίζουμε με `printf("%s\n", symb);`
- Αν ξεχάσουμε τον χαρακτήρα '\0' στο τέλος, η printf θα εμφανίσει "σκουπίδια" από τα επόμενα bytes μέχρι να βρεθεί κάποιο byte με τον μηδενικό χαρακτήρα. Με τη δήλωση:
`char *str = "A C-string";`
- ορίζουμε μια μεταβλητή τύπου δείκτη σε char με όνομα **str** που αρχικοποιείται στη συμβολοσειρά **"A C-string"**. Έτσι είναι νόμιμο να έχουμε τις εκχωρήσεις
`str = symb; ή str = "Another string";`

Πίνακες χαρακτήρων και συμβολοσειρές

- Ήδη γνωρίζουμε πώς μπορούμε να δημιουργήσουμε έναν πίνακα χαρακτήρων και να τον αρχικοποιήσουμε.
- Η αρχικοποίηση του πίνακα m1,

```
char m1[] = "Eisagwgh titlou bibliou: ";
```
- είναι ισοδύναμη με την κλασική αρχικοποίηση πινάκων:

```
char m1[] = {'E','i','s','a','g','w','g','h',' ','t','i','t','l','o','u',' ','b','i','b','l','i','o','u',' ','\0'};
```
- Το όνομα του πίνακα (m1) είναι ένας **σταθερός** δείκτης στο πρώτο στοιχείο του:

```
m1 == &m1[0], *m1 == m1[0] == 'E'
m1+1 == &m1[1], *(m1+1) == m1[1] == 'i'
...
m1+25 == &m1[25], *(m1+25) == m1[25] == '\0'
```
- Αφού ο m1 είναι σταθερός δείκτης, παραστάσεις όπως **m1++**; ή ***m1++**; Δεν επιτρέπονται. Όμως, παραστάσεις που αναφέρονται στο περιεχόμενο της θέσης μνήμης όπως η **(*m1)++**; επιτρέπονται.

Δείκτες σε char και συμβολοσειρές

- Σε αντίθεση με τους σταθερούς δείκτες που αντιστοιχούν σε πίνακες χαρακτήρων, μπορούμε να ορίσουμε και μεταβλητούς δείκτες σε char τους οποίους αρχικοποιούμε σε συμβολοσειρές όπως κάνουμε και με την αρχικοποίηση των τυπικών μεταβλητών.
- Έστω, **char *s1 = "Onoma Syggrafea: ", *s2;**
- Ο s1 δείχνει στον πρώτο χαρακτήρα της σταθερής συμβολοσειράς, δηλαδή στον 'O'. Όμως, τώρα επιτρέπεται να αυξήσουμε την τιμή ή να εκχωρήσουμε κάποια άλλη διεύθυνση στον δείκτη όπως στα παρακάτω παραδείγματα:

```
s2 = s1+5; /* Τώρα ο s2 δείχνει στο έκτο στοιχείο */
++s1; /* Τώρα ο s1 δείχνει στο επόμενο στοιχείο */
```
- Επίσης, επιτρέπεται το παρακάτω τμήμα κώδικα που εμφανίζει τη συμβολοσειρά s1:

```
while(*s1 != '\0')
    putchar(*s1++); /* Εμφανίζει τον χαρακτήρα στον οποίο δείχνει ο s1 και μετά αυξάνει τον s1 */
```
- **Προσοχή!!** Μεταβάλλοντας τον δείκτη χάνουμε και την πρόσβαση στην αρχή της συμβολοσειράς ...

Αναλυτικός προσδιορισμός αποθηκευτικού χώρου

- Αντί να αφήσουμε τον μεταγλωττιστή να μετρήσει το πλήθος των χαρακτήρων μιας συμβολοσειράς για να δεσμεύσει κατάλληλο χώρο στη μνήμη, όπως, για παράδειγμα, με τη δήλωση,

```
char str1[] = "This is a string";
```

| T | h | i | s | | | | | | | | | | | | |

θα μπορούσαμε να προσδιορίσουμε εμείς οι ίδιοι το πλήθος των χαρακτήρων που θα δεσμευθούν για την αποθήκευση της συμβολοσειράς, αρκεί να φροντίσουμε να είναι μεγαλύτερο τουλάχιστον κατά ένα byte για την αποθήκευση του τερματικού χαρακτήρα '\0'. Όλα τα επιπλέον bytes θα αρχικοποιηθούν στο '\0'.

- Για παράδειγμα, η δήλωση,

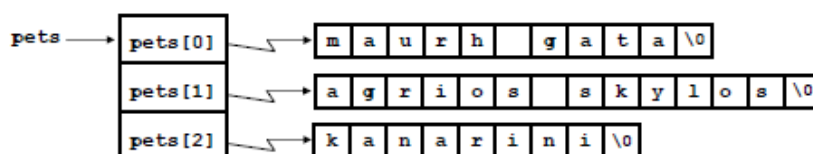
```
char str2[20] = "This is a string";
```

| T | h | i | s | | | | | | | | | | | | | | | | |

θα δεσμεύσει 20 bytes μνήμης για την αποθήκευση των 16 χαρακτήρων. Οι υπόλοιπες θέσεις μνήμης θα αρχικοποιηθούν στον χαρακτήρα '\0' όπως φαίνεται στο παρακάτω σχήμα. Επίσης, στο σχήμα παρουσιάζεται και η πρώτη περίπτωση (str1) για σύγκριση.

Πίνακες συμβολοσειρών

- Ας εξετάσουμε τη δήλωση του παρακάτω πίνακα συμβολοσειρών:
`char *pets[3] = {"maurh gata", "agrios skylos", "kanarini"};`
- Το όνομα **pets** είναι ένας πίνακας τριών δεικτών σε συμβολοσειρές. Φυσικά, η κάθε συμβολοσειρά είναι ένας πίνακας χαρακτήρων και συνεπώς έχουμε τρεις δείκτες σε πίνακες χαρακτήρων.
- Ο πρώτος δείκτης είναι ο **pets[0]** και δείχνει στη συμβολοσειρά "maurh gata", ο δεύτερος είναι ο **pets[1]** και δείχνει στη συμβολοσειρά "agrios skylos" και ο τρίτος είναι ο **pets[2]** και δείχνει στη συμβολοσειρά "kanarini".
- Σχηματικά έχουμε τρεις συνεχόμενες θέσεις μνήμης (ο πίνακας 3 στοιχείων) για την αποθήκευση των δεικτών με τον καθένα να δείχνει σε συμβολοσειρές διαφορετικού μήκους:



ΕΙΣΟΔΟΣ/ΕΞΟΔΟΣ ΣΥΜΒΟΛΟΣΕΙΡΩΝ

Η είσοδος μιας συμβολοσειράς προϋποθέτει ότι έχουμε δεσμεύσει αρκετό χώρο στη μνήμη που να την "χωράει". Ας σχολιάσουμε το ακόλουθο παράδειγμα:

```
#include <stdio.h>
#define MAXLEN 81 /* μέγιστο μήκος συμβολοσειράς + 1 */
int main() {
    char name1[MAXLEN], *name2;
    printf("Please enter your name: ");
    scanf("%s", name1); /* Δεν γίνεται έλεγχος αν χωράει στη name1 */
    printf("\n%s, please enter your father's name: ", name1);
    scanf("%s", name2); /* Μπορεί να αποθηκευθεί πάνω σε δεδομένα */
    ...
    return 0;
}
```

ΠΡΟΣΟΧΗ! Είναι ευθύνη του προγραμματιστή η δέσμευση αρκετής μνήμης για την αποθήκευση μιας συμβολοσειράς. Αν εξαντληθεί ο χώρος που δεσμευθήκε ή αν δεν είχε δεσμευθεί καθόλου (π.χ. όπως με την name2) υπάρχει κίνδυνος να γραφεί μνήμη που έχει δεσμευθεί για άλλα δεδομένα με, συνήθως, καταστροφικά αποτελέσματα.

Δυναμική Δέσμευση Μνήμης: η malloc()

- Εκτός από τη στατική δέσμευση μνήμης, μπορούμε να χρησιμοποιήσουμε και τη συνάρτηση βιβλιοθήκης malloc() για δυναμική δέσμευση μνήμης όταν κριθεί αναγκαίο.
- Για παράδειγμα, θα μπορούσαμε να χρησιμοποιήσουμε το **name1** ως pointer σε char και πριν τη scanf() να δεσμεύσουμε μνήμη (π.χ. 81 bytes) στην οποία θα δείχνει ο pointer name1 ως εξής:

```
#include <stdlib.h>
...
name1 = (char *) malloc(81);
```

- Επειδή το default της malloc() (ορίζεται στη βιβλιοθήκη stdlib.h) είναι να επιστρέφει pointer σε void (void *), χρειάζεται μετατροπή του τύπου δεδομένων σε (char *). Αν δεσμεύαμε χώρο για αποθήκευση, π.χ. δεδομένων τύπου int, τότε θα έπρεπε να φροντίσουμε για την μετατροπή του pointer στον τύπο (int *). Παράδειγμα:

```
int *pint;
pint = (int *) malloc(N*sizeof(int));
```

Η συνάρτηση εισόδου scanf() [1/3]

- Η scanf() διαβάζει συμβολοσειρές με το προσδιοριστικό μετατροπής %s. Η scanf() **σταματάει το διάβασμα** και προσθέτει το '\0' όταν συναντήσει κάποιον από τους "λευκούς" χαρακτήρες (space ' ', tab '\t', ή newline '\n') ενώ, επίσης, αγνοεί και τυχόν λευκούς χαρακτήρες στην αρχή. Δηλαδή, **στην ουσία, η scanf() διαβάζει μια λέξη**. Παράδειγμα:

```
#include <stdio.h>
#include <string.h> /* Περιέχει συναρτήσεις χειρισμού strings */
#define MAXLEN 81

int main() {
    char name1[MAXLEN], name2[MAXLEN];
    printf("Please enter your name: ");
    scanf("%s",name1);
    printf("\n%s, please enter your father's name: ",name1);
    scanf("%s",name2);
    printf("%s has %d characters\n",name1,strlen(name1));
    printf("%s has %d characters\n",name2,strlen(name2));
    return 0;
}
```

```
Please enter your name: Costas Vassiliou[enter]
Costas, please enter your father's name: Costas has 6 characters
Vassiliou has 9 characters
```

Η συνάρτηση εισόδου scanf() [2/3]

- Μόλις η scanf συναντήσει το space αμέσως μετά την πρώτη λέξη "Costas" επιστρέφει τη συμβολοσειρά Costas στον πίνακα χαρακτήρων name1 (βάζοντας και το '\0' στο τέλος) **και αφήνει τους υπόλοιπους χαρακτήρες (" Vassiliou\n") στο buffer εισόδου**. Αυτούς τους χαρακτήρες βρίσκει η δεύτερη scanf στο buffer με αποτέλεσμα να θέσει τη συμβολοσειρά "Vassiliou" στον πίνακα χαρακτήρων name2. Για να αδειάσουμε το buffer εισόδου πριν την εισαγωγή συμβολοσειράς μπορούμε να χρησιμοποιήσουμε την **fflush(stdin)**; ή τον βρόχο **while (getchar() != '\n');**

```
...
printf("Please enter your name: ");
scanf("%s",name1);
fflush(stdin); /* Empty buffer */
printf("\n%s, please enter your father's name: ",name1);
scanf("%s",name2);
printf("%s has %d characters\n",name1,strlen(name1));
printf("%s has %d characters\n",name2,strlen(name2));
...
```

```
Please enter your name: Costas Vassiliou[enter]
Costas, please enter your father's name: Giorgos[enter]
Costas has 6 characters
Giorgos has 7 characters
```


Η συνάρτηση εισόδου scanf() [3/3]

- Η scanf() επιστρέφει έναν ακέραιο που ισούται με το πλήθος των δεδομένων (οποιοδήποτε τύπου) που διαβάστηκαν. Αν συναντήσει τον ειδικό χαρακτήρα EOF (ορισμένος στην stdio.h ως **#define EOF (-1)**) επιστρέφει τον EOF. Παράδειγμα:

```
...
int val, nr_of_items;
char name1[20], name2[20];
...
nr_of_items = scanf("%d %s %s",&val,name1,name2);
printf("nr_of_items = %d\n",nr_of_items);
```

10 Hello World[enter]

nr_of_items = 3

Η συνάρτηση εισόδου gets() [1/3]

- Η συνάρτηση αυτή είναι πολύ βολική για αλληλεπιδραστικά προγράμματα καθώς διαβάζει όλους τους χαρακτήρες που υπάρχουν στο buffer εισόδου συμπεριλαμβανομένων και των λευκών χαρακτήρων.
- Η gets() διαβάζει χαρακτήρες μέχρι να συναντήσει τον χαρακτήρα νέας γραμμής '\n'. Στη συνέχεια, σχηματίζει μια συμβολοσειρά με τους χαρακτήρες που διάβασε (εκτός του '\n'), προσθέτει στο τέλος τον μηδενικό χαρακτήρα '\0' και δίνει τη συμβολοσειρά στην καλούσα συνάρτηση.
- Η μορφή gets() έχει ένα όρισμα τύπου δείκτη σε char για τη δεικτοδότηση της συμβολοσειράς. Παράδειγμα:

```
...
char titlos[512]; /* Δέσμευση μνήμης */
printf("Dwse titlo bibliou:\n");
gets(titlos); /* Eisagwgh symboloseiras */
printf("Titlos: \"%s\"\n",titlos);
...
```

Dwse titlo bibliou:

The C Programming Language[enter]

Titlos: "The C Programming Language"

Η συνάρτηση εισόδου gets() [2/3]

- Ένα άλλο χαρακτηριστικό της gets() είναι ότι επιστρέφει και δείκτη στη συμβολοσειρά που διαβάζει. Για παράδειγμα, το παρακάτω πρόγραμμα θα έχει την ίδια έξοδο με το προηγούμενο:

```
#include <stdio.h>
int main() {
    char titlos[512];
    char *str, *gets();

    printf("Dwse titlo bibliou:\n");
    str = gets(titlos); /* Eisagwgh symboloseiras */
    printf("Titlos: \"%s\"\n",str);
    return 0;
}
```

- Η δήλωση **char *gets();** μας λέει ότι η συνάρτηση gets() είναι τύπου δείκτη σε char.
- *Σημείωση:* Όταν δεν χρησιμοποιούμε την τιμή που επιστρέφει, όπως στο προηγούμενο πρόγραμμα, δεν χρειάζεται να κάνουμε την αντίστοιχη διακήρυξη.

Η συνάρτηση εισόδου gets() [3/3]

- Και ένα τελευταίο σχόλιο για την gets(): Αν όλα πάνε καλά, τότε η gets() επιστρέφει τη συμβολοσειρά που πληκτρολογήσαμε (ακόμη και αν δώσαμε την κενή συμβολοσειρά «» πατώντας αμέσως το [enter]).
- Αν όμως κάτι δεν πάει καλά ή αν πληκτρολογήσουμε τον ειδικό χαρακτήρα End-Of-File (EOF) που μπορεί να είναι και συνδυασμός χαρακτήρων σε κάποια συστήματα (π.χ. CTRL-D για το UNIX, CTRL-Z ή CTRL-C για Windows), τότε επιστρέφει ως τιμή τη μηδενική (NULL) διεύθυνση. Αυτό μας επιτρέπει να κάνουμε έλεγχο, αν όλα πήγαν καλά με το διάβασμα της συμβολοσειράς, της μορφής:

```
if(gets(name) != NULL)
...
- ή
while(gets(name) != NULL)
...
```

όπου η σταθερά NULL (μηδενική διεύθυνση) είναι ορισμένη στη βιβλιοθήκη stdio.h ως:

```
#define NULL 0
```

Η συνάρτηση εξόδου puts() [1/2]

- Η συνάρτηση αυτή έχει ένα όρισμα τύπου δείκτη σε συμβολοσειρά. Μόλις η puts() συναντήσει τον μηδενικό χαρακτήρα, τον αντικαθιστά με τον '\n' και εμφανίζει τη συμβολοσειρά στην οθόνη. Παράδειγμα:

```
#include <stdio.h>
#define STR "Constant string with #define"
int main() {
    char str1[] = "String as array of chars";
    char *str2 = "Pointer to a string";
    puts("A typical constant string");
    puts(STR); puts(str1); puts(str2);
    puts(&str1[10]); puts(str2+11);
    return 0;
}
```

A typical constant string
Constant string with #define
String as array of chars
Pointer to a string
array of chars
a string

Η συνάρτηση εξόδου puts() [2/2]

Ας υποθέσουμε ότι θέλουμε να γράψουμε τη δική μας συνάρτηση myputs() που εκτός από την εμφάνιση της συμβολοσειράς να επιστρέφει και το πλήθος των χαρακτήρων που εμφανίζει. Αυτό μπορεί να γίνει εύκολα, όπως φαίνεται στο παρακάτω παράδειγμα.

```
#include <stdio.h>

int myputs(char *str) {
    int count = 0;
    while(*str != '\0') {
        putchar(*str++); count++;
    }
    putchar('\n');
    return(count);
}

int main() {
    char flowers[80];
    puts("Give me the names of your favorite flowers: ");
    gets(flowers);
    printf("Nr of chars = %d\n", myputs(flowers));
    return 0;
}
```

Give me the names of your favorite flowers:
roses, lilies and tulips[enter]
roses, lilies and tulips
Nr of chars = 24

Η συνάρτηση εξόδου printf()

- Η συνάρτηση αυτή δεν είναι τόσο βολική όσο η puts() όταν θέλουμε να εμφανίσουμε μια συμβολοσειρά. Για παράδειγμα, η εμφάνιση της συμβολοσειράς string γίνεται με την

```
printf("%s\n",string);
```

γράφοντας περισσότερο κώδικα απ'ότι με την **puts(string);**

- Επιπλέον, με την puts() δεν χρειάζεται να περιλαμβάνουμε το '\n'. Αν όμως θέλουμε να συνδυάσουμε συμβολοσειρές σε μια γραμμή εκτύπωσης τότε αυτό μπορεί να γίνει εύκολα και απλά με την printf(), όπως, για παράδειγμα:

```
printf("%s: %s, %s\n", name, str1, str2);
```

ΣΥΝΑΡΤΗΣΕΙΣ ΧΕΙΡΙΣΜΟΥ ΣΥΜΒΟΛΟΣΕΙΡΩΝ

- Οι περισσότερες βιβλιοθήκες της C (π.χ. η string.h) περιλαμβάνουν και συναρτήσεις χειρισμού συμβολοσειρών. Οι πιο συνηθισμένες από αυτές είναι οι εξής:

Η συνάρτηση strlen()

```
int strlen(char *str);
```

- Η συνάρτηση αυτή (string length) μετράει τους χαρακτήρες μέχρι να συναντήσει τον χαρακτήρα '\0' και επιστρέφει το μήκος της συμβολοσειράς.

Η συνάρτηση strcat()

```
char *strcat(char *str1, char *str2);
```

- Η συνάρτηση αυτή (string concatenation) συνενώνει τις δύο συμβολοσειρές που δέχεται στα ορίσματά της ως εξής: ένα αντίγραφο της δεύτερης συμβολοσειράς προστίθεται στο τέλος της πρώτης αλλάζοντάς την. Η δεύτερη συμβολοσειρά δεν τροποποιείται.
- ΠΡΟΣΟΧΗ! Πριν την κλήση της strcat() πρέπει να γίνεται έλεγχος από το πρόγραμμα αν η δεύτερη συμβολοσειρά χωράει στον δεσμευμένο χώρο της πρώτης. Παράδειγμα (υποθέτει ότι η name έχει το μικρό όνομα και η lastname το επώνυμο):

```
strcat(name, " ");
```

```
strcat(name, lastname);
```

Η συνάρτηση strcmp()

```
int strcmp(char *str1, char *str2);
```

Η συνάρτηση αυτή (string comparison) συγκρίνει δύο συμβολοσειρές και επιστρέφει 0 αν οι συμβολοσειρές είναι ίδιες. Αν είναι διαφορετικές τότε, σε κάποιες περιπτώσεις, η strcmp() εντοπίζει τη θέση στις συμβολοσειρές που για πρώτη φορά υπάρχει διαφορά χαρακτήρων και επιστρέφει τη διαφορά των ASCII κωδικών τους.

Σε άλλες υλοποιήσεις της strcmp() (π.χ. της Dev-C++), επιστρέφονται μόνο τρεις τιμές: 0 για ίδιες συμβολοσειρές, 1 αν είναι διαφορετικές με τον χαρακτήρα της πρώτης να έχει μεγαλύτερο ASCII κωδικό από τον αντίστοιχο χαρακτήρα της δεύτερης και -1 στην αντίθετη περίπτωση. Παράδειγμα:

```
#include <stdio.h>
#include <string.h>
int main() {
    printf("%d\n", strcmp("A", "A"));
    printf("%d\n", strcmp("A", "B"));
    printf("%d\n", strcmp("B", "A"));
    printf("%d\n", strcmp("Dennis Ritchie", "Dennis Richie"));
    return 0;
}
```

```
0
-1
1
1
```

Η συνάρτηση strcpy()

```
char *strcpy(char *str1, char *str2);
```

Η συνάρτηση αυτή (string copy) αντιγράφει το περιεχόμενο της δεύτερης συμβολοσειράς στην πρώτη και επιστρέφει δείκτη στην πρώτη.

Θα πρέπει να έχουμε δεσμεύσει αρκετό χώρο για την πρώτη συμβολοσειρά με δήλωση της μορφής,

```
char string1[SIZE];
```

και όχι της μορφής

```
char *string1;
```

που δημιουργεί δείκτη σε συμβολοσειρά αλλά δεν δεσμεύει μνήμη.

Χρήσιμες μακροεντολές και συναρτήσεις χαρακτήρων

Πολλές φορές χρειαζόμαστε να ελέγξουμε αν κάποιοι χαρακτήρες που πιθανόν πληκτρολόγησε ο χρήστης ή που διαβάσαμε από κάποιο αρχείο είναι αλφαβητικοί, αριθμητικοί, σημεία στίξης, κεφαλαία γράμματα, κ.λπ. Η βιβλιοθήκη `ctype.h` της C (περιλάβετε την οδηγία του προεπεξεργαστή: `#include <ctype.h>`) μας επιτρέπει να χρησιμοποιήσουμε ένα πλήθος μακροεντολών που έχουν ως παράμετρο έναν απλό χαρακτήρα `c` και επιστρέφουν `TRUE` αν ο έλεγχος ικανοποιείται και `FALSE` στην αντίθετη περίπτωση.

`isalnum(c)` `TRUE` αν ο `c` είναι αλφαριθμητικός χαρακτήρας.
`isalpha(c)` `TRUE` αν ο `c` είναι αλφαβητικός χαρακτήρας.
`isascii(c)` `TRUE` αν ο `c` είναι ASCII χαρακτήρας (0–0177).
`iscntrl(c)` `TRUE` αν ο `c` είναι χαρακτήρας ελέγχου (0–0378 ή 01778).
`isdigit(c)` `TRUE` αν ο `c` είναι αριθμητικό ψηφίο (digit).
`isgraph(c)` `TRUE` αν ο `c` είναι εκτυπώσιμος χαρακτήρας (0418–01768).
`islower(c)` `TRUE` αν ο `c` είναι πεζός (lower-case) χαρακτήρας.
`isprint(c)` `TRUE` αν ο `c` είναι εκτυπώσιμος χαρακτήρας ή διάστημα.
`ispunct(c)` `TRUE` αν ο `c` είναι χαρακτήρας στίξης.
`isspace(c)` `TRUE` αν ο `c` είναι χαρακτήρας διαστήματος.
`isupper(c)` `TRUE` αν ο `c` είναι κεφαλαίος (upper-case) χαρακτήρας.
`isxdigit(c)` `TRUE` αν ο `c` είναι δεκαεξαδικό ψηφίο.

Επίσης, παρέχονται και δύο συναρτήσεις για μετατροπές χαρακτήρων.

`int tolower(c)` Επιστρέφει τον πεζό αλφαβητικό χαρακτήρα που αντιστοιχεί στον `c`.
`int toupper(c)` Επιστρέφει τον κεφαλαίο αλφαβητικό χαρακτήρα που αντιστοιχεί στον `c`.