ASCII - κωδικοί

- **ASCII (American Standard Code for Information Interchange)**
- **Η ASCII** κωδικοποίηση αποτελεί το πλέον χρησιμοποιούμενο πρότυπο για την ανταλλαγή πληροφοριών και επικοινωνία μεταξύ συστημάτων επεξεργασίας δεδομένων, συμπεριλαμβανομένου και του Internet.
- Το αρχικό σύνολο περιελάμβανε 128 ASCII χαρακτήρες (βλ. **Πίνακα ASCII**) κωδικοποιημένοι με 7 bits συμπεριλαμβανομένων των αλφαβητικών (πεζών και κεφαλαίων) και αριθμητικών χαρακτήρων, ειδικών συμβόλων καθώς και άλλων μη εκτυπώσιμων χαρακτήρων.
- Σήμερα υπάρχουν επεκτάσεις του Πίνακα ASCII που περιλαμβάνουν και χαρακτήρες άλλων γλωσσών.

ΠΙΝΑΚΑΣ ASCII

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Χαρακτήρες στη C

- Ένας χαρακτήρας κωδικοποιείται με τον αντίστοιχο ASCII κωδικό του. Ο κωδικός αυτός είναι ακέραιος και επειδή αποτελείται από 7 bits θα είναι από 0 έως 127. Για παράδειγμα, ο χαρακτήρας 'A' αντιστοιχεί στον κωδικό 65 (στο δεκαδικό σύστημα):

```
printf("character: %c - ASCII code: %d\n",'A','A');
```

```
character: A - ASCII code: 65
```

- Επειδή οι ASCII κωδικοί είναι ακέραιοι αριθμοί, μπορούμε να κάνουμε αριθμητικές εκχωρήσεις σε μεταβλητές τύπου char (υπενθυμίζουμε ότι ο τύπος char ανήκει στους ακέραιους τύπους δεδομένων) ή ακόμη και αριθμητικές πράξεις:

```
char ch1, ch2;
ch1 = 70;          /* DECIMAL 70 = HEX 46 = OCTAL 106 */
ch2 = ++ch1 + 5;
printf("ch1 = %c [%d], ch2 = %c [%d]\n",ch1,ch1,ch2,ch2);
ch1 = G [71], ch2 = L [76]
```

- Προσδιορισμός ASCII κωδικών στο οκταδικό σύστημα

```
char ch1 = '\07', ch2 = '\106';
printf("Attention!'%c': ",ch1);
printf("The ASCII code of %c in octal is %o\n",ch2,ch2);
Attention!<beep>: The ASCII code of F in octal is 106
```

ΣΥΜΒΟΛΟΣΕΙΡΕΣ (STRINGS)

- Μια συμβολοσειρά είναι μια ακολουθία ενός ή περισσοτέρων 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 0_{10} ή 0_8 ή $000\ 0000_2$ και είναι εντελώς διαφορετικός από το ψηφίο 0 με ASCII κωδικό 48_{10} ή 60_8 ή $110\ 0000_2$ ('0' ≡ '\60').

ΣΥΜΒΟΛΟΣΕΙΡΕΣ (STRINGS)

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)

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";
```

Σταθερές συμβολοσειρές

Ό,τι περικλείεται σε διπλά εισαγωγικά, αναγνωρίζεται από τον compiler ως μια σταθερή συμβολοσειρά. Οι N χαρακτήρες της συμβολοσειράς καθώς και ο '\0' αποθηκεύονται σε (N + 1) συνεχόμενες θέσεις μνήμης του ενός byte. Σταθερές συμβολοσειρές έχουμε επανειλημμένως χρησιμοποιήσει στα προηγούμενα προγράμματα ως ορίσματα της printf().

Σημειώστε ότι μπορούμε να ορίσουμε σταθερές συμβολοσειρές και με την **#define**. Π.χ.

```
#define STRING "This is a string"
int main() { printf("%s\n",STRING); return 0;}
```

με έξοδο: `This is a string`

Μια σταθερή συμβολοσειρά στην ουσία δρα ως δείκτης στην περιοχή που έχει αποθηκευθεί, όπως ακριβώς δρα και το όνομα ενός πίνακα. Εξετάστε το ακόλουθο πρόγραμμα:

```
int main() /* Συμβολοσειρές ως pointers */
{
  printf("%u, %u, %c\n","string","string",*"string");
  return 0;
}
```

ορίζεται μια φορά (π.χ. Dev-C++) ή πολλές φορές στη μνήμη, ανάλογα με το σύστημα

με έξοδο: `4206592, 4206592, s`

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:

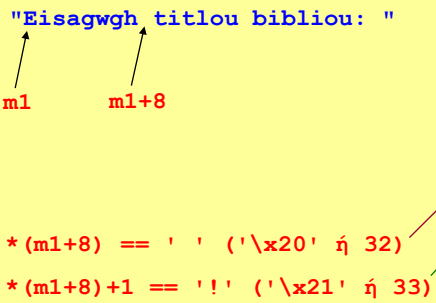
```
m1 == &m1[0],  *m1 == m1[0] == 'E'
m1+1 == &m1[1], *(m1+1) == m1[1] == 'i'
...
m1+25 == &m1[25], *(m1+25) == m1[25] == '\0'
```

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.

Ερώτηση: Τι θα εμφανίσει η `printf("%c\n",*(m1+8)+1);` ??

Απάντηση: Τον επόμενο χαρακτήρα (βλ. πίνακα ASCII) του κενού διαστήματος που είναι ο `'!'` [το *(m1+8) είναι ο κενός χαρακτήρας με ASCII κωδικό 32_{10} ή 20_{16}].

ΠΙΝΑΚΑΣ ASCII



	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

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 'Ο'. 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 */
```

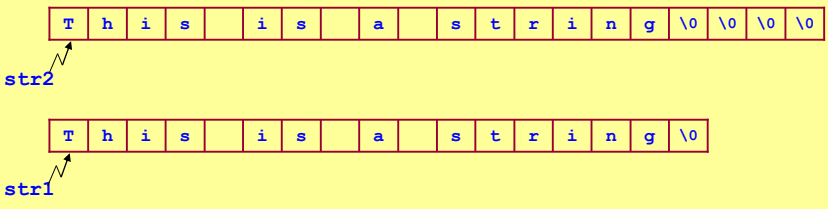
Προσοχή!! *Μεταβάλλοντας τον δείκτη χάνουμε και την πρόσβαση στην αρχή της συμβολοσειράς …*

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";`] 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";
```

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.

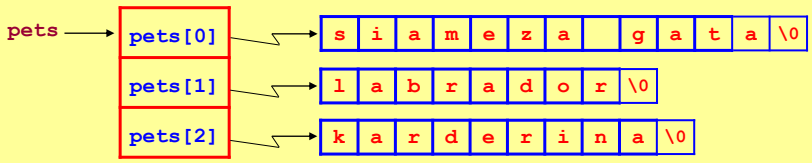


Πίνακες συμβολοσειρών

Ας εξετάσουμε τη δήλωση του παρακάτω πίνακα συμβολοσειρών:

```
char *pets[3] = {"siameza gata", "labrador", "karderina"};
```

Το όνομα `pets` είναι ένας πίνακας τριών δεικτών σε συμβολοσειρές. Φυσικά, η κάθε συμβολοσειρά είναι ένας πίνακας χαρακτήρων και συνεπώς έχουμε τρείς 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 τρείς συνεχόμενες θέσεις μνήμης (ο πίνακας 3 στοιχείων) για την αποθήκευση των δεικτών με τον καθένα να δείχνει σε συμβολοσειρές διαφορετικού μήκους:

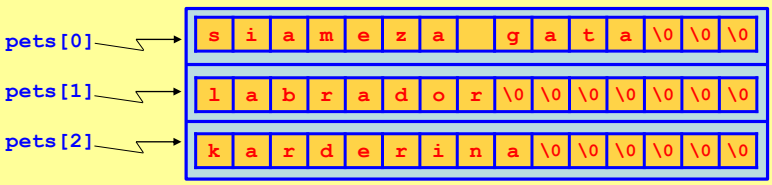


Πίνακες συμβολοσειρών

Δήλωση 2-Δ πίνακα τύπου char για αποθήκευση συμβολοσειρών:

```
char pets[3][15] = {"siameza gata", "labrador", "karderina"};
```

Ο `pets` στο παραπάνω παράδειγμα είναι ένας 2-Δ πίνακας για αποθήκευση τριών συμβολοσειρών το πολύ 14 χαρακτήρων. Όπως και προηγουμένως, ο `pets[i]` θα δείχνει στην `i` συμβολοσειρά. Σχηματικά η αποθήκευση των συμβολοσειρών στον πίνακα θα είναι όπως στο παρακάτω παράδειγμα:



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 δεσμεύθηκε 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()

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 scanf() (συνέχεια …)

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: `while (getchar() != '\n');`

```
  . . .
  printf("Please enter your name: ");
  scanf("%s",name1);

  while (getchar() != '\n');       /* 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 scanf() (συνέχεια …)

Η scanf() επιστρέφει έναν ακέραιο που ισούται με το πλήθος των δεδομένων (οποιουδήποτε 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()

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 gets() (συνέχεια ...)

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;

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

Η gets() (συνέχεια ...)

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()

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 puts() (συνέχεια ...)

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 διακήρυξη 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 (0–127).
iscntrl(c)   TRUE αν ο c είναι χαρακτήρας ελέγχου.
isdigit(c)   TRUE αν ο c είναι αριθμητικό ψηφίο (digit).
isgraph(c)   TRUE αν ο c είναι εκτυπώσιμος χαρακτήρας.
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.