



**Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών**

Εισαγωγή στον Προγραμματισμό

Η γλώσσα προγραμματισμού C

Ο τύπος του χαρακτήρα και ο ASCII Κώδικας

Η C όπως κάθε γλώσσα προγραμματισμού έχει το δικό της **αλφάβητο**, το οποίο χρησιμοποιεί για το σχηματισμό λέξεων που αποτελούν το λεξιλόγιο της.
Καθώς η Μηχανή μπορεί να αποθηκεύσει στην μνήμη της μόνο αριθμούς δημιουργήθηκαν πίνακες αντιστοίχισης χαρακτήρων σε αριθμούς, όπως είναι ο πίνακας ASCII ...

κάθε
χαρακτήρας
έχει ένα
μοναδικό
αριθμητικό
κωδικό

55	067	37	00110111	7	7		Seven
56	070	38	00111000	8	8		Eight
57	071	39	00111001	9	9		Nine
58	072	3A	00111010	:	:	:	Colon
59	073	3B	00111011	;	;	;	Semicolon
60	074	3C	00111100	<	<	<	Less than (or op
61	075	3D	00111101	=	=	=	Equals
62	076	3E	00111110	>	>	>	Greater than (or
63	077	3F	00111111	?	?	?	Question mark
64	100	40	01000000	@	@	@	At sign
65	101	41	01000001	A	A		Uppercase A
66	102	42	01000010	B	B		Uppercase B
67	103	43	01000011	C	C		Uppercase C

<https://www.ascii-code.com/ASCII>

Μεταβλητή τύπου χαρακτήρα

```
char ch;
```

δηλώνει μεταβλητή με όνομα ch ως τύπου χαρακτήρα έχει ως αποτέλεσμα τη δέσμευση χώρου στην μνήμη για την αποθήκευση ενός χαρακτήρα

```
char ch1 = 'A' ;
```

δηλώνει μεταβλητή με όνομα ch1 και της αποδίδει ως αρχική τιμή τον χαρακτήρα A

```
void putchar(char ch) ;
```

Δήλωση της συνάρτησης putchar της τυπικής βιβλιοθήκης για την εμφάνιση στην κύρια έξοδο ενός χαρακτήρα

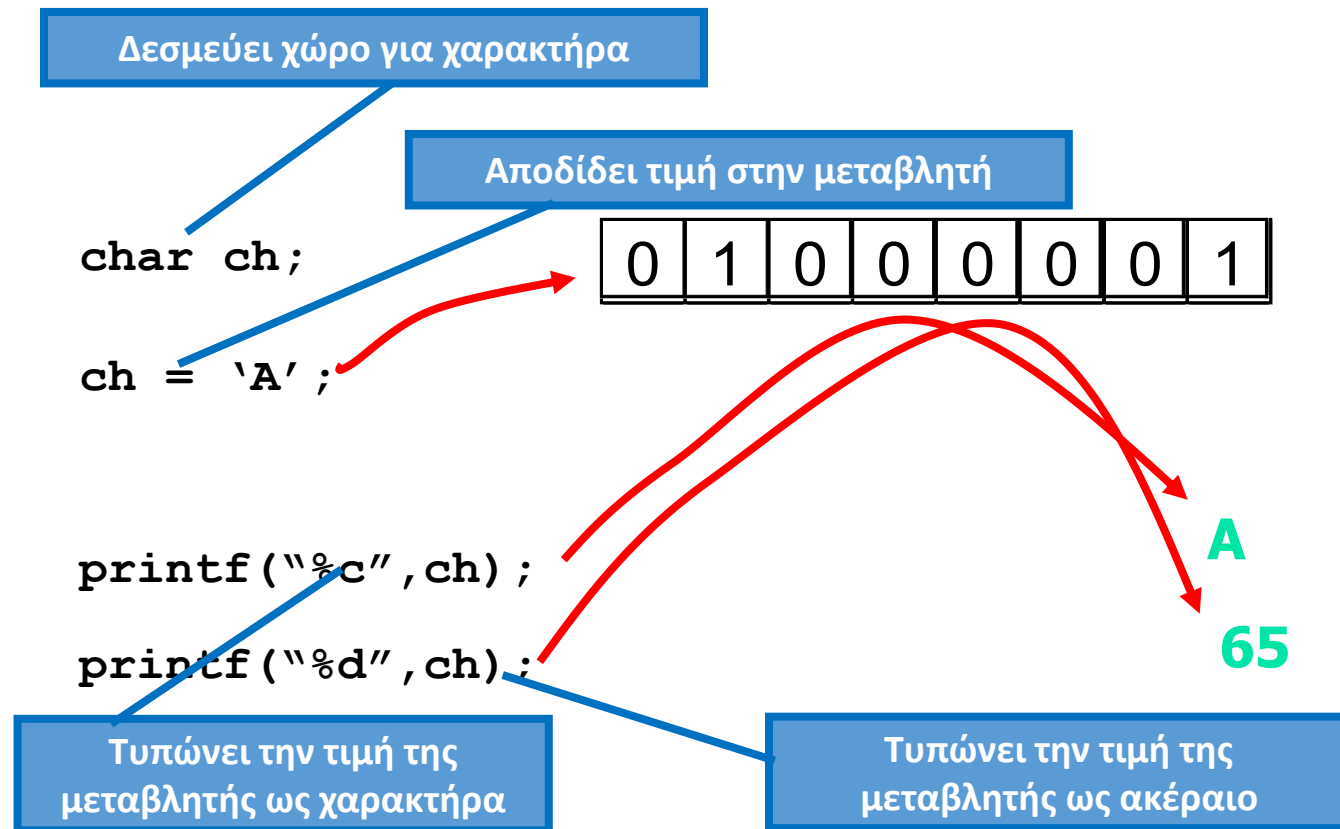
```
printf("ch1=%c \n", ch1) ;
```

Παράδειγμα χρήσης της printf για εμφάνιση στην κύρια έξοδο ενός χαρακτήρα

```
printf("ch1 as decimal =%d \n", ch1) ;
```

Παράδειγμα χρήσης της printf για εμφάνιση στην κύρια έξοδο του ascii κωδικού ενός χαρακτήρα

Ο τύπος του χαρακτήρα και ο ASCII Κώδικας



Πρόγραμμα: Εμφάνιση χαρακτήρα στη κύρια έξοδο

```
mainV1.c X
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "../i2p.h"
4
5  char chl='A';
6  char tab = '\t';
7  char newLine = '\n';
8
9  int main()
10 {
11     printf("CharHandlingV1!\n");
12     printf("chl=%c. \n", chl);
13     printf("chl as decimal =%d. \n", chl);
14
15     printf("tab=%c. \n", tab);
16     printf("tab as decimal =%d. \n", tab);
17
18     printf("newLine=%c. \n", newLine);
19     printf("newLine as decimal =%d. \n", newLine);
20
21     return 0;
22 }
```

C:\Code\I2P\Chapter6\CharHi

CharHandlingV1!
chl=A.
chl as decimal =65.
tab= .
tab as decimal =9.
newLine=
.
newLine as decimal =10.

Είσοδος χαρακτήρα – getchar

```
char getchar(void) ;
```

Δήλωση της συνάρτησης getchar της τυπικής βιβλιοθήκης για είσοδο χαρακτήρα από την κύρια έξοδο

```
ch = getchar() ;
```

Παράδειγμα χρήσης της getchar για είσοδο από την κύρια είσοδο ενός χαρακτήρα και απόδοσης του ως τιμής σε μεταβλητή χαρακτήρα

Συμβολοσειρά (String) και αλφαριθμητικό

Συμβολοσειρά **είναι** μια ακολουθία συμβόλων που ανήκουν σε ένα πεπερασμένο σύνολο, το οποίο ονομάζεται αλφάβητο.

Αν η συμβολοσειρά αποτελείται μόνο από αλφαριθμητικούς χαρακτήρες και αριθμούς τότε ονομάζεται αλφαριθμητικό.

Η C δεν διαθέτει τύπο για συμβολοσειρά, αλλά χειρίζεται τις **συμβολοσειρές ως πίνακες χαρακτήρων**.

Το αλφάβητο της γλώσσας C σύμφωνα με το ANSI (American Standard Code for Information Interchange) πρότυπο αποτελείται από 96 χαρακτήρες.

a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9
_ { } [] # () < > % : ; . ? * + - / ^ & | ~ ! = , \ " ' "

Συμβολοσειρά (String) ως πίνακας χαρακτήρων

Για να θεωρηθεί ένας πίνακας χαρακτήρων συμβολοσειρά θα πρέπει μετά τον τελευταίο χαρακτήρα του να υπάρχει ο χαρακτήρας με ascii κωδικό 0, ο οποίος είναι γνωστός ως **χαρακτήρας null** ('\\0').

```
char name[20] = {'h','e','l','l','o','\\0'};
```

```
char name[] = {'h','e','l','l','o','\\0'};  
char name[] = "hello";
```

Δεσμεύουν χώρο για τον αριθμό των **χαρακτήρων** συν ένα για τον χαρακτήρα '\\0'. Στην περίπτωση μας 6 (5+1).

Αναφορά
στοιχείου

Μνήμη

Διεύθυνση

name[0]

h

F000

name[1]

e

F001

name[2]

l

F002

l

F003

o

F004

\\0

F005

F006

....

name[19]

F019

Το name είναι η διεύθυνση στο πρώτο στοιχείο του πίνακα

Είσοδος και έξοδος συμβολοσειράς

- Για **έξοδο συμβολοσειράς** χρησιμοποιούμε τη γενική συνάρτηση εξόδου **printf** με τον **προσδιοριστή %s**.
- Η συνάρτηση τυπώνει στην κύρια έξοδο τους χαρακτήρες της συμβολοσειράς μέχρι να συναντήσει τον χαρακτήρα null.

```
char message[20] = {'h','e','l','l','o','\0'};  
printf("message is %s\n", message);
```

Για **είσοδο συμβολοσειράς** δίνουμε συναρτήσεις :

scanf (της τυπικής βιβλιοθήκης)
gets
fgets

Αλφαριθμητικά - Πίνακες

- Αλφαριθμητικό
 - ακολουθία αλφαβητικών και αριθμητικών χαρακτήρων (π.χ. Όνομα χρήστη, τίτλος βιβλίου, ISBN βιβλίου κλπ)
- Pascal -> ειδικός τύπος (string)
- C -> πίνακας χαρακτήρων με τελευταίο στοιχείο το μηδενικό χαρακτήρα ('\0')

Αλφαριθμητικά

- Εκτύπωση

```
printf("Hello");
```

```
printf("Ο αριθμός είναι %s\n", str);
```

- Εισαγωγή

```
scanf("%s", str); ή scanf("%s", &str[0]);
```

διαβάζει μέχρι το πρώτο κενό

προσοχή στην υπέρβαση του μεγέθους

- Συμβολοσειρές

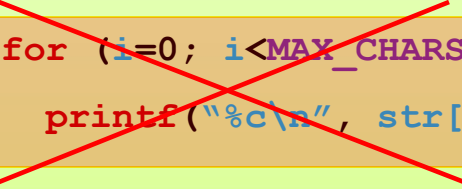
- Μια συμβολοσειρά στη C είναι ένας πίνακας χαρακτήρων.
- Το μήκος μιας συμβολοσειράς καθορίζεται από έναν τερματικό μηδενικό χαρακτήρα: `'\0'`.
- Έτσι, μια συμβολοσειρά με τα περιεχόμενα, ας πούμε, `"timer"` έχει έξι χαρακτήρες: `'t'` , `'i'` , `'m'` , `'e'` , `'r'` και τον τερματικό μηδενικό (`'\0'`) χαρακτήρα.
- Ο τερματικός μηδενικός χαρακτήρας έχει την τιμή στο ASCII μηδέν.

Παράδειγμα

```
#include <stdio.h>

#define MAX_CHARS 80


int main()
{
    char str[MAX_CHARS];
    int i;
    printf("Δώσε αριθμητικό:");
    scanf("%s", str);
    for (i=0; i<MAX_CHARS; i++)
        printf("%c\n", str[i]);
}
```



```
#include <stdio.h>

#define MAX_CHARS 80

int main()
{
    char str[MAX_CHARS];
    int i;
    printf("Δώσε αριθμητικό:");
    scanf("%s", str);
    i = 0;
    while (str[i] != '\0')
        printf("%c\n", str[i++]);
}
```



Συναρτήσεις Αλφαριθμητικών

□ <stdio.h>

```
char *gets(char *s)    // διαβάζει string από το stdin στο s
                        // και επιστρέφει το s

int puts(const char *s) // γράφει το string s στο stdout

Char * fgets(char*s, int n, FILE *fstream); // διαβάζει το πολύ n-1
//χαρακτήρες σταματώντας αν συναντήσει χαρακτήρα νέας γραμμής
```

□ <string.h>

```
int  strlen(cs)
int  strcmp(cs,ct)                int  strncmp(cs, ct, count);
char *strcpy(dest,source)        char *strncpy(dest, source, count)
char *strcat(destination, source)
```

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

```
int main()
```

```
{
```

```
    char x[20];
```

```
    printf("Give: ");
```

```
    scanf("%19[^\n]s", x);
```

```
    puts(x);
```

```
}
```

<https://learn.microsoft.com/en-us/cpp/c-runtime-library/scanf-width-specification?view=msvc-170>

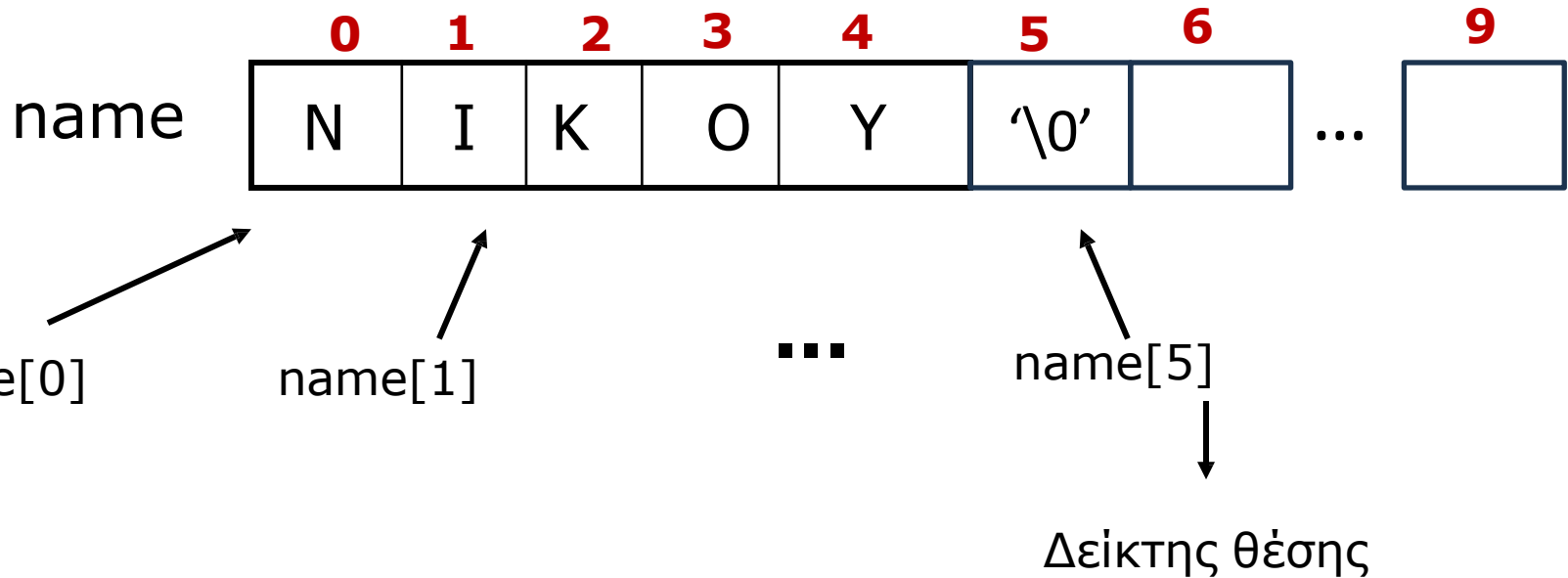
Για να διαβάσετε συμβολοσειρές που δεν περιορίζονται από χαρακτήρες κενών, μπορείτε να αντικαταστήσετε με ένα σύνολο χαρακτήρων σε αγκύλες ([]) τον χαρακτήρα τύπου s (συμβολοσειρά). Το σύνολο των χαρακτήρων σε αγκύλες αναφέρεται ως *συμβολοσειρά ελέγχου*. Το αντίστοιχο πεδίο εισόδου διαβάζεται μέχρι τον πρώτο χαρακτήρα που δεν εμφανίζεται στη συμβολοσειρά ελέγχου. Εάν ο πρώτος χαρακτήρας στο σύνολο είναι ένα καπέλο (^), το αποτέλεσμα αντιστρέφεται: το πεδίο εισόδου διαβάζεται μέχρι τον πρώτο χαρακτήρα που εμφανίζεται στο υπόλοιπο σύνολο χαρακτήρων. Και οι δύο %[a-z] και %[z-a] ερμηνεύονται ως ισοδύναμες με %[abcde...z]. Είναι μια κοινή επέκταση της συνάρτησης scanf, αλλά δεν απαιτείται από το πρότυπο C.

Δείκτες και αλφαριθμητικά (εμφάνιση με printf())

Μπορούμε να εμφανίσουμε όποιο τμήμα του αλφαριθμητικού θέλουμε ξεκινώντας από την αντίστοιχη θέση. Για παράδειγμα για να εκτυπώσουμε στο αλφαριθμητικό name (char name[]) από την τέταρτη θέση και μετά, αρκεί να γράψουμε printf("%s", name+3) ή printf("%s", &name[3]);

```
char name[N]="ΝΙΚΟΥ";  
/* char name[N]="ΝΙΚΟΥ";  
   char name[]="ΝΙΚΟΥ"; */  
  
int len, i;  
len=strlen(name);  
for (i=0; i<len; i++)  
    // printf("%s\n", name+i);  
    printf("%s\n", &name[i]);  
  
return 0;
```

```
char name[N]="NIKOY";
```



Δείκτες - Αλφαριθμητικά (1)

Δήλωση δείκτη αλφαριθμητικού

char * <όνομα – δείκτη>;

π.χ. char * pmsg;

Ανάθεση σε δείκτη αλφαριθμητικού

<όνομα-δείκτη> = < αλφαριθμητικό >;

π.χ. pmsg = "Today is Thursday";

Δείκτες - Αλφαριθμητικά (2)

Προσοχή στις διαφορές

`char msg[] = "Today is Thursday";` (πίνακας χαρακτήρων)

`char * pmsg = "Today is Thursday";` (δείκτης σε πίνακα, μπορεί να πάρει άλλη τιμή)

`char * msg[18];` (πίνακας δεικτών χαρακτήρα)

`msg[1]` -> 2ος δείκτης

`* (msg[1])` -> ο 1ος χαρακτήρας του δεύτερου δείκτη

Η συνάρτηση strlen()

Η συνάρτηση `strlen()` δηλώνεται στο αρχείο `string.h` και επιστρέφει τον αριθμό των χαρακτήρων που περιέχει ένα αλφαριθμητικό, χωρίς να μετράει τον τερματικό χαρακτήρα (`'\0'`)

Η συνάρτηση δηλώνεται ως:

```
int strlen(char str[]);
```

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

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

```
#define N 100
```

```
int main()
```

```
{
```

```
    int len;
```

```
    char name[N];
```

```
    printf("Give name: ");
```

```
    scanf("%s", name);
```

```
    len=strlen(name);
```

```
    printf("The length is %d\n", len);
```

```
    return 0;
```

```
}
```

Η συνάρτηση `strcpy()` (1/2)



Ας ξεκινήσουμε με ένα πολύ συνηθισμένο λάθος, αυτό της χρήσης του τελεστή ανάθεσης = για την αντιγραφή αλφαριθμητικών, π.χ.:

```
char str[10];  
str = "something"; /* Λάθος */
```

Ένας σωστός τρόπος να αντιγράψουμε ένα αλφαριθμητικό είναι με τη συνάρτηση `strcpy()` που δηλώνεται στο αρχείο `string.h` και χρησιμοποιείται για την αντιγραφή ενός αλφαριθμητικού σε μία άλλη θέση μνήμης (string copy)

Η συνάρτηση `strcpy()` δέχεται σαν παραμέτρους δύο δείκτες και αντιγράφει το αλφαριθμητικό στο οποίο δείχνει ο δεύτερος δείκτης (source), συμπεριλαμβανομένου του τερματικού χαρακτήρα, στη μνήμη που δείχνει ο πρώτος δείκτης (dest)

Το πρωτότυπό της δηλώνεται ως εξής:

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

Όταν αντιγραφεί και ο τερματικός χαρακτήρας, η συνάρτηση `strcpy()` τερματίζει και επιστρέφει τον δείκτη `dest` (δηλαδή δείχνει στη διεύθυνση μνήμης που δείχνει και ο δείκτης `dest`)

Η συνάρτηση strcpy() (2/2)

Π.χ. η παρακάτω εντολή αντιγράφει το αλφαριθμητικό "something" στον πίνακα str

```
char str[100];  
strcpy(str, "something");
```

Αν συνεχίσουμε το παραπάνω με μία δεύτερη strcpy() , π.χ. όπως αυτή που φαίνεται παρακάτω, ποια θα είναι η έξοδος?

```
strcpy(str, "new");  
printf("%c\n", str[6]);
```

Αφού η strcpy() τερματίζει όταν αντιγραφεί ο τερματικός χαρακτήρας, αλλάζουν οι τέσσερις πρώτες θέσεις και οι υπόλοιπες παραμένουν το ίδιο, δηλαδή εμφανίζεται το i

Παρατηρήσεις (1/2)



Επειδή η `strcpy()` δεν ελέγχει αν η μνήμη προορισμού στην οποία θα αντιγραφεί το αλφαριθμητικό χωράει όλους τους χαρακτήρες του, πρέπει να έχετε εξασφαλίσει ότι το μέγεθός της θα είναι αρκετά μεγάλο, ώστε να αποφευχθεί η υπερεγγραφή μνήμης

Με άλλα λόγια, να προσέχετε, ώστε το μέγεθος της μνήμης που έχει δεσμευτεί για το 1ο αλφαριθμητικό να είναι αρκετά μεγάλο για να χωράει όλους τους χαρακτήρες του 2ου αλφαριθμητικού.

Αν δεν είναι, τότε οι πλεονάζοντες χαρακτήρες θα εγγραφούν σε μη δεσμευμένη μνήμη, το οποίο μπορεί να προκαλέσει την δυσλειτουργία του προγράμματος

Π.χ. στο επόμενο πρόγραμμα, αφού το μέγεθος του πίνακα `str` είναι μικρότερο από ό,τι πρέπει, η `strcpy()` υπερεγγράφει τα δεδομένα που υπάρχουν στη μνήμη μετά το όριο του πίνακα, με αποτέλεσμα την πιθανή δυσλειτουργία του προγράμματος

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char c = 'a', str[10];

    strcpy(str, "Longer text. The program may crash");
    printf("%s %c\n", str, c);
    return 0;
}
```


Παρατηρήσεις (2/2)



Και ναι, είναι **πολύ σοβαρό λάθος** να μεταβιβάσουμε έναν δείκτη που δεν έχει αρχικοποιηθεί στην `strcpy()`



Δηλαδή, **μεγάλη προσοχή**, πριν την αντιγραφή του αλφαριθμητικού **πρέπει να έχει προηγηθεί** η δέσμευση της αντίστοιχης μνήμης

Για παράδειγμα, το επόμενο πρόγραμμα δεν θα εκτελεστεί σωστά, γιατί **δεν έχει δεσμευτεί μνήμη για την αποθήκευση** του αλφαριθμητικού

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char *str;

    strcpy(str, "something");
    printf("%s\n", str);
    return 0;
}
```

Παραδείγματα (I)

Γράψτε ένα πρόγραμμα το οποίο να διαβάζει ένα αλφαριθμητικό μέχρι 100 χαρακτήρες και να το αντιγράφει σε έναν πίνακα χαρακτήρων με χρήση της συνάρτησης `strcpy()`

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char str1[100], str2[100];

    printf("Enter text: ");
    fgets(str2, sizeof(str2), stdin);

    strcpy(str1, str2);
    printf("Copied text: %s\n", str1);
    return 0;
}
```

Παραδείγματα (II)

Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char str1[10],str2[10];
    printf("%s\n",strcpy(str1,strcpy(str2,"test")));
    return 0;
}
```

Έξοδος: test

Η συνάρτηση `strncpy()`

Η συνάρτηση `strncpy()` δηλώνεται στο αρχείο `string.h` και χρησιμοποιείται για την αντιγραφή ενός συγκεκριμένου πλήθους χαρακτήρων ενός αλφαριθμητικού σε μία άλλη θέση μνήμης (`string copy n chars`)

Η συνάρτηση `strncpy()` είναι παρόμοια με τη συνάρτηση `strcpy()` με τη διαφορά ότι **δέχεται μία επιπλέον παράμετρο**, που είναι **το πλήθος των χαρακτήρων που θα αντιγραφούν**

Το πρωτότυπό της δηλώνεται ως εξής:

```
char *strncpy(char *dest, const char *source, size_t count);
```

Εάν η τιμή της παραμέτρου `count` είναι μικρότερη από το πλήθος των χαρακτήρων του αλφαριθμητικού στο οποίο δείχνει ο `source` δείκτης, τότε δεν προστίθεται ο τερματικός χαρακτήρας `'\0'` στο τέλος της μνήμης που δείχνει ο `dest` δείκτης

Εάν είναι μεγαλύτερη, τότε προστίθενται τερματικοί χαρακτήρες `'\0'` στο τέλος της μνήμης που δείχνει ο `dest` δείκτης μέχρι να συμπληρωθεί ο αριθμός `count`

Αφού η `strncpy()` καθορίζει το μέγιστο πλήθος των χαρακτήρων που θα αντιγραφούν **είναι ασφαλέστερη** από την `strcpy()`

Παράδειγμα

Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char str1[] = "Old text", str2[] = "New", str3[] = "Get";

    strncpy(str1, str2, 3);
    printf("%s\n", str1);

    strncpy(str1, str3, 5);
    printf("%s\n", str1);
    return 0;
}
```

Έξοδος: New text
Get

Η συνάρτηση `strcat()`

Η συνάρτηση `strcat()` δηλώνεται στο αρχείο `string.h` και χρησιμοποιείται για τη συνένωση ενός αλφαριθμητικού με ένα άλλο (string concatenate)

Η συνάρτηση `strcat()` δέχεται σαν παραμέτρους δύο δείκτες και προσθέτει το αλφαριθμητικό στο οποίο δείχνει ο δείκτης `source` στο τέλος της μνήμης που δείχνει ο δείκτης `dest`

Ο τερματικός χαρακτήρας `'\0'` προστίθεται αυτόματα στο τέλος του νέου αλφαριθμητικού

Το πρωτότυπό της δηλώνεται ως εξής:

```
char *strcat(char *dest, const char *source);
```

Η συνάρτηση `strcat()` επιστρέφει τον δείκτη `dest`, ο οποίος δείχνει στη μνήμη που περιέχει τα ενωμένα αλφαριθμητικά

Παρατηρήσεις

Επειδή η `strcat()` δεν ελέγχει αν η μνήμη στην οποία θα προστεθεί το αλφαριθμητικό χωράει όλους τους χαρακτήρες του, **το μέγεθος της μνήμης που έχει δεσμευτεί για το πρώτο αλφαριθμητικό πρέπει να είναι αρκετά μεγάλο για να χωράει τους χαρακτήρες και των δύο αλφαριθμητικών**

Αν δεν είναι, οι πλεονάζοντες χαρακτήρες θα εγγραφούν σε θέσεις μνήμης μετά από το επιτρεπτό όριο, υπερεγγράφοντας τα δεδομένα που υπάρχουν εκεί, π.χ.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    char str[20] = "example";

    strcat(str, "not available memory");
    printf("The merged text is: %s\n", str);
    return 0;
}
```

Προφανώς, ισχύει επίσης κι η αντίστοιχη παρατήρηση της συνάρτησης `strcpy()` για τη δέσμευση μνήμης πριν τη συνένωση αλφαριθμητικών, δηλαδή πριν την αντιγραφή του αλφαριθμητικού **πρέπει να έχει προηγηθεί** η δέσμευση της αντίστοιχης μνήμης

Παράδειγμα

Γράψτε ένα πρόγραμμα το οποίο να διαβάζει δύο αλφαριθμητικά μέχρι 100 χαρακτήρες, να τα ενώνει σε ένα τρίτο αλφαριθμητικό με χρήση της συνάρτησης `strcat()` και να το εμφανίζει στην οθόνη

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

int read_text(char str[], int size, int flag);

int main(void)
{
    char str1[100], str2[100], str3[200] = {0}; /* Ο
    τερματικός χαρακτήρας πρέπει να υπάρχει αποθηκευμένος στον str3,
    ώστε να εκτελεστεί σωστά η πρώτη strcat(). */

    printf("Enter first text: ");
    read_text(str1, sizeof(str1), 1);

    printf("Enter second text: ");
    read_text(str2, sizeof(str2), 1);

    strcat(str3, str1); /* Το αλφαριθμητικό που περιέχεται
    στον str1 θα προστεθεί στον πίνακα str3. */
    strcat(str3, str2);
    printf("The merged text is: %s\n", str3);
    return 0;
}
```


Η συνάρτηση strcmp () (1/3)

Η συνάρτηση strcmp () δηλώνεται στο αρχείο string.h και χρησιμοποιείται για τη σύγκριση αλφαριθμητικών (string compare)

Η συνάρτηση strcmp () δέχεται σαν παραμέτρους δύο δείκτες και συγκρίνει το αλφαριθμητικό στο οποίο δείχνει ο δείκτης str1 με το αλφαριθμητικό στο οποίο δείχνει ο δείκτης str2

Το πρωτότυπό της δηλώνεται ως εξής:

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

Αν τα δύο αλφαριθμητικά είναι ακριβώς ίδια, τότε η συνάρτηση strcmp () επιστρέφει την τιμή μηδέν (0)

Αν το πρώτο αλφαριθμητικό είναι μικρότερο από το δεύτερο, τότε επιστρέφει μία αρνητική τιμή, ενώ αν είναι μεγαλύτερο επιστρέφει μία θετική τιμή

Η συνάρτηση `strcmp()` (2/3)

Ένα αλφαριθμητικό **θεωρείται μικρότερο** από κάποιο άλλο αν ισχύει ένα από τα παρακάτω:

- α) οι πρώτοι n χαρακτήρες των αλφαριθμητικών είναι ίδιοι, αλλά η **τιμή του πρώτου μη κοινού χαρακτήρα** στο πρώτο αλφαριθμητικό **είναι μικρότερη** από την τιμή του αντίστοιχου χαρακτήρα στο δεύτερο αλφαριθμητικό
- β) οι **χαρακτήρες** τους είναι οι **ίδιοι**, αλλά το **μήκος** του στο πρώτο αλφαριθμητικό είναι **μικρότερο**

Δείτε παρακάτω, θεωρώντας ότι η σύγκριση των αλφαριθμητικών βασίζεται στις ASCII τιμές που έχουν οι χαρακτήρες τους (θυμηθείτε ότι στον ASCII κώδικα τα κεφαλαία γράμματα έχουν μικρότερη τιμή από τα αντίστοιχα πεζά)

Επομένως, η εντολή:

```
strcmp("tHE", "the");
```

επιστρέφει μία αρνητική τιμή, ενώ η εντολή:

```
strcmp("yes", "Yes");
```

επιστρέφει μία θετική τιμή, αντίστοιχα

Η συνάρτηση strcmp () (3/3)

Συνεχίζοντας, η εντολή:

```
strcmp("w", "many");
```

επιστρέφει μία θετική τιμή (αφού η ASCII τιμή του πρώτου μη κοινού χαρακτήρα 'w' είναι μεγαλύτερη από την αντίστοιχη του 'm'),

ενώ η εντολή:

```
strcmp("some", "something");
```

επιστρέφει μία αρνητική τιμή, αντίστοιχα, διότι μπορεί οι πρώτοι τέσσερις χαρακτήρες των δύο αλφαριθμητικών να είναι ίδιοι, αλλά το μήκος του πρώτου αλφαριθμητικού είναι μικρότερο από το μήκος του δεύτερου

Παρατηρήσεις



Ένα πολύ συνηθισμένο λάθος είναι η χρήση του τελεστή `==` για τη σύγκριση αλφαριθμητικών, π.χ. δείτε το παρακάτω:

```
#include <stdio.h>
int main(void)
{
    char str1[] = "test", str2[] = "test";

    (str1 == str2) ? printf("One\n") : printf("Two\n");
    return 0;
}
```

Δεδομένου ότι τα ονόματα των δύο πινάκων χρησιμοποιούνται ως δείκτες, η έκφραση `str1 == str2` **ελέγχει αν οι δείκτες δείχνουν στην ίδια διεύθυνση και όχι αν τα περιεχόμενα των πινάκων είναι ίδια**

Δείχνουν οι `str1` και `str2` στην ίδια διεύθυνση?

Φυσικά και όχι!

Οι πίνακες `str1` και `str2` μπορεί να έχουν το ίδιο περιεχόμενο, αλλά σίγουρα είναι αποθηκευμένοι σε διαφορετικές διευθύνσεις μνήμης

Επομένως, το πρόγραμμα εμφανίζει `Two`

Η συνάρτηση `strncmp()`

Η συνάρτηση `strncmp()` είναι παρόμοια με τη `strcmp()`, δηλώνεται κι αυτή στο αρχείο `string.h` και χρησιμοποιείται για να συγκρίνει ένα συγκεκριμένο πλήθος χαρακτήρων (`string compare n chars`)

Το πρωτότυπό της δηλώνεται ως εξής:

```
int strncmp(const char *str1, const char *str2, int count);
```

Η παράμετρος `count` δηλώνει το πλήθος των χαρακτήρων που θα συγκριθούν

Παράδειγμα

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

int read_text(char str[], int size, int flag);

int main(void)
{
    char str1[100], str2[100];
    int k;

    printf("Enter first text: ");
    read_text(str1, sizeof(str1), 1);

    printf("Enter second text: ");
    read_text(str2, sizeof(str2), 1);

    k = strcmp(str1, str2);
    /* We could omit the declaration of k and write:
    if(strcmp(str1, str2) == 0) */
    if(k == 0)
        printf("Same texts\n");
    else
    {
        printf("Different texts\n");
        if(strncmp(str1, str2, 3) == 0)
            printf("But the first 3 chars are the same\n");
    }
    return 0;
}
```

Γράψτε ένα πρόγραμμα το οποίο να διαβάζει δύο αλφαριθμητικά μέχρι 100 χαρακτήρες και να τα συγκρίνει με χρήση της συνάρτησης `strcmp()`.

Αν τα αλφαριθμητικά είναι διαφορετικά, να συγκρίνει τους 3 πρώτους χαρακτήρες τους με χρήση της συνάρτησης `strncmp()` και να εμφανίζει κατάλληλο μήνυμα.

Διδιάστατοι πίνακες και αλφαριθμητικά (I)

Οι διδιάστατοι πίνακες χρησιμοποιούνται πολύ συχνά για την αποθήκευση αλφαριθμητικών

Π.χ., με την εντολή:

```
char str[3][40];
```

δηλώνεται ο πίνακας `str`, ο οποίος περιέχει 3 γραμμές και σε κάθε γραμμή του πίνακα μπορεί να αποθηκευτεί ένα αλφαριθμητικό μέχρι 40 χαρακτήρες

Μπορούμε να αποθηκεύσουμε κυριολεκτικά αλφαριθμητικά σε έναν διδιάστατο πίνακα ταυτόχρονα με τη δήλωσή του

Π.χ. με τη δήλωση:

```
char str[3][40] = {"The", "There", "Three"};
```

οι χαρακτήρες του `"The"` αποθηκεύονται στην πρώτη γραμμή του πίνακα `str`, του `"There"` στη δεύτερη και του `"Three"` στην τρίτη

Παράδειγμα (2)

```
void strcpy(char *s, char *t)
{
    int i=0;
    while ((s[i]=t[i])!='\0')
        i++;
}
```

```
void strcpy(char *s, char *t)
{
    while ((*s=*t) != '\0') { s++; t++;}
}
```

```
void strcpy(char *s, char *t)
{
    while ((* s++=*t++) != '\0') ;
}
```


Παράδειγμα (3)

```
int strcmp(char *s, char *t)
{
    int i=0;
    for (i=0; (s[i]==t[i])&& s[i]!='\0'; i++);
    return s[i]-t[i];
}
```

```
int strcmp(char *s, char *t)
{
    for (; (*s==*t) && *s!='\0'; s++, t++);
    return *s-*t;
}
```