


Αλφαριθμητικά

Αλφαριθμητικά - Εισαγωγή

- Ένα αλφαριθμητικό (*string*) είναι μία ακολουθία χαρακτήρων, η οποία πρέπει να τελειώνει με έναν ειδικό χαρακτήρα, ο οποίος είναι ο χαρακτήρας `'\0'`
- Ο χαρακτήρας αυτός (`'\0'`) ονομάζεται **τερματικός χαρακτήρας** (*null character*) και **οριοθετεί το τέλος του αλφαριθμητικού**
-  Προσοχή στο ότι ο τερματικός χαρακτήρας `'\0'` και ο χαρακτήρας `'0'` είναι δύο διαφορετικοί χαρακτήρες
- Η ASCII τιμή του τερματικού χαρακτήρα είναι 0 (μηδέν), ενώ του μηδενικού ψηφίου είναι 48

Κυριολεκτικά Αλφαριθμητικά

- Μία ακολουθία χαρακτήρων που περιέχεται μέσα σε **διπλά εισαγωγικά** ονομάζεται **κυριολεκτικό αλφαριθμητικό** και αποθηκεύεται στη μνήμη σαν να ήταν πίνακας χαρακτήρων
- Για παράδειγμα, αν ο μεταγλωττιστής συναντήσει στο πρόγραμμα το αλφαριθμητικό "message" δεσμεύει για αυτό οκτώ θέσεις μνήμης, ώστε να αποθηκεύσει τους επτά χαρακτήρες του και τον τερματικό χαρακτήρα ('\0 ')
- Αφού ένα κυριολεκτικό αλφαριθμητικό αποθηκεύεται σαν πίνακας χαρακτήρων, μπορούμε να το χρησιμοποιήσουμε και σαν «δείκτη σε χαρακτήρα», δηλαδή δείκτη με τύπο `char*`

Αποθήκευση Αλφαριθμητικών

- Η αποθήκευση αλφαριθμητικών γίνεται σε μεταβλητές που δηλώνονται σαν πίνακες χαρακτήρων
- Π.χ. η πρόταση:

```
char str[10];
```

δηλώνει έναν πίνακα `str` με στοιχεία 10 χαρακτήρες

ΣΗΜΑΝΤΙΚΗ ΠΑΡΑΤΗΡΗΣΗ

- Στην πραγματικότητα, στον πίνακα `str` επιτρέπεται να αποθηκεύσουμε μέχρι και 9 χαρακτήρες και όχι 10, γιατί μία θέση δεσμεύεται για τον τερματικό χαρακτήρα `'\0'`



Γενικά, για να αποθηκεύσουμε ένα αλφαριθμητικό που μπορεί να έχει μέχρι και N χαρακτήρες θα πρέπει να δηλώσουμε έναν πίνακα με N+1 θέσεις χαρακτήρων

Αποθήκευση Αλφαριθμητικών με τη Δήλωση (I)

Α' Τρόπος

- Τη δήλωση του πίνακα την ακολουθεί ο τελεστής = και οι χαρακτήρες του αλφαριθμητικού διαχωρίζονται με τον τελεστή κόμμα (,) μέσα σε άγκιστρα { }

ΠΑΡΑΔΕΙΓΜΑ

```
char str[8] = {'m','e','s','s','a','g','e','\0'};
```

- Στο παραπάνω παράδειγμα η τιμή του `str[0]` γίνεται 'm', η τιμή του `str[1]` γίνεται 'e', η τιμή του `str[2]` γίνεται 's', κ.ο.κ.
- Προφανώς, η τιμή του τελευταίου στοιχείου του πίνακα `str` (δηλ. του `str[7]`) γίνεται '\0'

Αποθήκευση Αλφαριθμητικών με τη Δήλωση (II)

Β' Τρόπος

- Παραπλήσιος με τον Α' Τρόπο, αλλά πιο βολικός, γιατί το αλφαριθμητικό μπαίνει ανάμεσα σε διπλά εισαγωγικά ""

- ΠΑΡΑΔΕΙΓΜΑ

```
char str[8] = "message";
```

- Όπως και προηγουμένως, η τιμή του κάθε στοιχείου του πίνακα `str` γίνεται ίση με τον αντίστοιχο χαρακτήρα του αλφαριθμητικού
- Ο μεταγλωττιστής προσθέτει αυτόματα τον τερματικό χαρακτήρα στο τέλος του αλφαριθμητικού, άρα η τιμή του τελευταίου στοιχείου του πίνακα (δηλ. του `str[7]`) γίνεται `'\0'`

Αποθήκευση Αλφαριθμητικών με τη Δήλωση (III)

Γ' Τρόπος

- Το κύριο μειονέκτημα των 2 προηγούμενων τρόπων ήταν ότι ο προγραμματιστής έπρεπε να μετρήσει τον αριθμό των χαρακτήρων του αλφαριθμητικού, να προσθέσει και τον τερματικό χαρακτήρα, ώστε να υπολογίσει τη διάσταση του πίνακα
- Προφανώς, αυτή η διαδικασία είναι χρονοβόρα, ιδίως στην περίπτωση που το αλφαριθμητικό έχει πολλούς χαρακτήρες, και - εκτός από αυτό - είναι πολύ πιθανό να προκύψει και λάθος στο μέτρημα
- Ο τρίτος και **πιο ευέλικτος τρόπος** είναι να μην δηλωθεί η διάσταση του πίνακα, ώστε να την υπολογίσει αυτόματα ο μεταγλωττιστής
- ΠΑΡΑΔΕΙΓΜΑ

```
char str[] = "message";
```
- Σε αυτό το παράδειγμα ο μεταγλωττιστής δημιουργεί έναν πίνακα χαρακτήρων με τόσες θέσεις όσες χρειάζονται για να αποθηκευτούν στα στοιχεία του οι χαρακτήρες του αλφαριθμητικού "message" και ο τερματικός χαρακτήρας
- Με αυτόν τον τρόπο ο προγραμματιστής δεν χρειάζεται να μετρήσει το μήκος του αλφαριθμητικού για να υπολογίσει τη διάσταση του πίνακα

Παρατηρήσεις (I)

- Ο μεταγλωττιστής για να ανακαλύψει το τέλος ενός αλφαριθμητικού ψάχνει να βρει τον **τερματικό χαρακτήρα** (`'\0'`)
- Επομένως, κάθε αλφαριθμητικό πρέπει να τελειώνει με τον τερματικό χαρακτήρα `'\0'`
- Επίσης, ο τερματικός χαρακτήρας (`'\0'`) πρέπει να υπάρχει, για να μπορούν να λειτουργήσουν σωστά και οι αντίστοιχες συναρτήσεις χειρισμού αλφαριθμητικών (π.χ. `strlen()`, `strcpy()`, ...) τις οποίες θα εξετάσουμε παρακάτω

Παρατηρήσεις (II)

- Ένας τρόπος για να εξασφαλιστεί ότι ο τερματικός χαρακτήρας `'\0'` θα περιέχεται οπωσδήποτε σε έναν πίνακα χαρακτήρων, είναι να τεθεί στα στοιχεία του η τιμή `'\0'`, κατά τη δήλωσή του (δηλ. να γίνει ταυτόχρονη αρχικοποίηση με τη δήλωσή του)

Π.χ. με τη δήλωση:

```
char str[100] = {0};
```

όλα τα στοιχεία του πίνακα `str` γίνονται ίσα με την ASCII τιμή 0, η οποία αντιστοιχίζεται στον τερματικό χαρακτήρα `'\0'`

Παρατηρήσεις (III)

- Όπως με όλους τους πίνακες, αν το πλήθος των χαρακτήρων είναι μικρότερο από το μέγεθος του πίνακα, οι τιμές των υπολοίπων στοιχείων αρχικοποιούνται με μηδέν
- Αφού η ASCII τιμή του `'\0'` είναι μηδέν (0), σημαίνει ότι αρχικοποιούνται με `'\0'`
- Π.χ. με τη δήλωση:

```
char str[8] = "me";
```

το `str[0]` γίνεται `'m'`, το `str[1]` γίνεται `'e'` και τα υπόλοιπα στοιχεία (`str[2]`, `str[3]`, ..., `str[7]`) ίσα με `'\0'`

Παραδείγματα (I)

- Το παρακάτω πρόγραμμα αποθηκεύει το κυριολεκτικό αλφαριθμητικό "This is the text" σε έναν πίνακα χαρακτήρων. Υπάρχει κάποια αδυναμία ???

```
#include <stdio.h>
int main()
{
    char str[100] = "This is the text";
    return 0;
}
```

Η αδυναμία έγκειται στο ότι έχουν δεσμευτεί στη μνήμη 100 bytes ενώ αν η αποθήκευση του αλφαριθμητικού γινόταν ως:

```
char str[] = "This is the text";
```

θα δεσμεύονταν ακριβώς 17 bytes (όσα δηλ. οι χαρακτήρες του αλφαριθμητικού (16) συν τον τερματικό χαρακτήρα)

Παραδείγματα (II)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
    char str[] = "This is the text";
    printf("%d\n", sizeof(str));
    return 0;
}
```

Έξοδος: 17

Παραδείγματα (III)

- Ποια είναι η διαφορά μεταξύ των εκφράσεων `"a"` και `'a'` ???

`"a"` : Αλφαριθμητικό, πίνακας χαρακτήρων με δύο χαρακτήρες, τον χαρακτήρα `'a'` και τον τερματικό χαρακτήρα `'\0'`

`'a'` : απλώς, ο χαρακτήρας `'a'`

Παραδείγματα (IV)

- Υπάρχει προγραμματιστικό λάθος στο παρακάτω πρόγραμμα ???

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

    str2[4] = 'w';
    printf("%c\n", str1[0]);
    return 0;
}
```

Με την εντολή `char str1[] = "abc";` ο μεταγλωττιστής δημιουργεί έναν πίνακα τεσσάρων θέσεων, για να αποθηκεύσει τους χαρακτήρες 'a', 'b', 'c' και τον '\0' και παρόμοια τον πίνακα `str2` για να αποθηκεύσει τους χαρακτήρες 'e', 'f', 'g' και τον '\0'.

Το λάθος συμβαίνει όταν το πρόγραμμα επιχειρεί να αποθηκεύσει τον χαρακτήρα 'w' σε θέση του πίνακα `str2` που είναι **εκτός των ορίων του**.

Άρα, με την εντολή `str2[4] = 'w'` γίνεται **υπερεγγραφή μνήμης** και **αυθαίρετη αλλοίωση των δεδομένων** που είναι αποθηκευμένα εκεί.

Δηλαδή, το πρόγραμμα μπορεί να εμφανίσει 'a' μπορεί όμως και να εμφανίσει 'w' αν οι πίνακες `str1` και `str2` έχουν αποθηκευτεί σε διαδοχικές θέσεις μνήμης.

Εμφάνιση Αλφαριθμητικών με την `printf()` (I)

- Για την εμφάνιση ενός αλφαριθμητικού, μπορούμε να χρησιμοποιήσουμε τη γνωστή μας συνάρτηση `printf()`
- Για την εμφάνιση αλφαριθμητικού χρησιμοποιούμε το προσδιοριστικό `%s` στην εντολή `printf()` και έναν δείκτη στο αλφαριθμητικό
- Η συνάρτηση `printf()` εμφανίζει όλους τους χαρακτήρες από τον πρώτο χαρακτήρα στον οποίο δείχνει ο δείκτης μέχρι να συναντήσει τον τερματικό χαρακτήρα (`'\0'`)
- Στο επόμενο παράδειγμα χρησιμοποιείται το όνομα του πίνακα (αφού το όνομα ενός πίνακα είναι δείκτης στο πρώτο του στοιχείο)

```
#include <stdio.h>
int main()
{
    char str[] = "This is text";
    printf("%s\n", str);
    return 0;
}
```

Εμφάνιση Αλφαριθμητικών με την `printf()` (II)

- Προφανώς, μπορούμε να εμφανίσουμε οποιοδήποτε τμήμα του αλφαριθμητικού επιθυμούμε κάνοντας τον δείκτη να δείχνει στην ανάλογη θέση
- Για παράδειγμα, για να εμφανίσουμε στο προηγούμενο πρόγραμμα το τμήμα του αλφαριθμητικού από τον έκτο χαρακτήρα και μετά, δηλαδή το `is text`, θα γράφαμε:

```
#include <stdio.h>
int main()
{
    char str[] = "This is text";
    printf("%s\n", str+5);
    return 0;
}
```

ή ισοδύναμα:

```
#include <stdio.h>
int main()
{
    char str[] = "This is text";
    printf("%s\n", &str[5]);
    return 0;
}
```


Εμφάνιση Αλφαριθμητικών με την `puts()`

- Για την εμφάνιση ενός αλφαριθμητικού, μπορούμε να χρησιμοποιήσουμε και τη συνάρτηση `puts()`, η οποία λειτουργεί περίπου όπως και η `printf()`
- Η συνάρτηση `puts()` δεν χρειάζεται ως παράμετρο το προσδιοριστικό `%s` και απαιτεί ως μοναδική παράμετρο έναν δείκτη στην ακολουθία χαρακτήρων που επιθυμούμε να εμφανιστεί στην οθόνη και την εμφανίζει μέχρι να συναντήσει τον τερματικό χαρακτήρα (`'\0'`)
- Στο τέλος της ακολουθίας χαρακτήρων η `puts()` αντικαθιστά αυτόματα τον τερματικό χαρακτήρα (`'\0'`) με τον χαρακτήρα νέας γραμμής (`'\n'`)

```
#include <stdio.h>
int main()
{
    char str[] = "This is text";
    puts(str);
    return 0;
}
```

Παράδειγμα (I)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
    char str[100];

    str[0] = 'a';
    str[1] = 'b';
    printf("%s\n", str);
    return 0;
}
```

Πιθανή Έξοδος (επειδή δεν έχει
αρχικοποιηθεί ο πίνακας χαρακτήρων και άρα
δεν περιέχει τον τερματικό χαρακτήρα (' \0 ')):

ab | 9Tⁿ ▼

Παράδειγμα (II)

- Ποια είναι η έξοδος των παρακάτω προγραμμάτων ???

```
#include <stdio.h>
int main()
{
    char str[100] = "xy";

    str[0] = 'a';
    str[1] = 'b';
    printf("%s\n", str);
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    char str[100] = {0};

    str[0] = 'a';
    str[1] = 'b';
    printf("%s\n", str);
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    char str[100];

    str[0] = 'a';
    str[1] = 'b';
    str[2] = '\0';
    printf("%s\n", str);
    return 0;
}
```

Έξοδος: ab

Παράδειγμα (III)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
    char str[] = "Right'0'Wrong";
    printf("%s\n", str);
    return 0;
}
```

Απαντήσατε Right ???

Προσοχή!!! Το '0' που περιέχεται στο αλφαριθμητικό δεν αντιστοιχεί στον τερματικό χαρακτήρα ('\0'), αλλά στους εξής τρεις χαρακτήρες: το μονό εισαγωγικό, το μηδέν και ξανά το μονό εισαγωγικό.

Άρα, το πρόγραμμα εμφανίζει: Right'0'Wrong

Για να ήταν η απάντησή σας σωστή, θα έπρεπε να είχαμε δηλώσει το αλφαριθμητικό ως:

```
char str[] = "Right\0Wrong";
```

Δείκτες και Αλφαριθμητικά (I)

- Ένας εναλλακτικός τρόπος για να χειριστούμε ένα κυριολεκτικό αλφαριθμητικό είναι να δηλώσουμε έναν δείκτη που να δείχνει στον πρώτο χαρακτήρα του αλφαριθμητικού αυτού
- Παράδειγμα:

```
#include <stdio.h>
int main()
{
    char* ptr;
    ptr = "This is text";
    printf("First char = %c\n", *ptr);
    return 0;
}
```



Δείκτες και Αλφαριθμητικά (II)

- Δηλώνοντας έναν δείκτη σε ένα αλφαριθμητικό μπορούμε να το χειριστούμε με τον ίδιο τρόπο όπως αν το είχαμε αποθηκεύσει σε έναν πίνακα χαρακτήρων
- Δηλαδή με τις εντολές:

```
char *ptr = "This is text";
```

και

```
char ptr[] = "This is text";
```

μπορούμε να χειριστούμε το αλφαριθμητικό με τη χρήση του δείκτη `ptr`, γιατί και στη δεύτερη περίπτωση το `ptr` είναι επίσης δείκτης στη διεύθυνση του πρώτου χαρακτήρα του αλφαριθμητικού

- Και γιατί αυτό????
Θυμηθείτε ότι το όνομα ενός πίνακα (χωρίς τις αγκύλες) είναι δείκτης στο πρώτο στοιχείο του

Δείκτες και Αλφαριθμητικά (III)

- Τους **χαρακτήρες ενός αλφαριθμητικού** μπορούμε να τους χειριστούμε είτε χρησιμοποιώντας τη σημειογραφία δείκτη είτε χρησιμοποιώντας τη θέση τους στον πίνακα

- Παράδειγμα:

```
#include <stdio.h>
int main()
{
    int i;
    char* ptr;

    ptr = "This is text";
    for(i = 0; ptr[i] != '\0'; i++)
        printf("%c %c\n", *(ptr+i), ptr[i]);

    return 0;
}
```

- Παρατηρήστε ότι ο **for** βρόχος εκτελείται μέχρι να συναντήσουμε τον τερματικό χαρακτήρα **'\0'**

char *ptr vs. char ptr[]

```
char *ptr = "This is text";
```

```
char ptr[] = "This is text";
```

- 1) Αν έχουμε χρησιμοποιήσει τον 2^ο τρόπο δήλωσης, δηλαδή:

```
char ptr[] = "This is text";
```

μετά τη δήλωση και την αρχικοποίηση αυτή, επιτρέπεται να αποθηκεύσουμε στον πίνακα `ptr` ένα άλλο αλφαριθμητικό, λαμβάνοντας όμως υπόψιν **τον περιορισμό ότι το μήκος του νέου αλφαριθμητικού δεν θα πρέπει να υπερβαίνει το μέγεθος του πίνακα, που ισούται με το πλήθος των χαρακτήρων του αλφαριθμητικού αρχικοποίησης (δηλ. του "This is text") αυξημένο κατά ένα (για τον τερματικό χαρακτήρα `'\0'`)**

char *ptr vs. char

ptr[]

```
char *ptr = "This is text";
```

```
char ptr[] = "This is text";
```

2) **Αντίθετα**, αν έχουμε χρησιμοποιήσει τον 1ο τρόπο δήλωσης, δηλαδή:

```
char *ptr = "This is text";
```

μετά τη δήλωση και την αρχικοποίηση αυτή, επιτρέπεται να κάνουμε τον δείκτη ptr να δείξει σε ένα άλλο αλφαριθμητικό, ακόμα και αν αυτό έχει περισσότερους χαρακτήρες

Παράδειγμα:

```
#include <stdio.h>
int main()
{
    char* ptr = "First text";
    ptr = "This is a new text";
    printf("First char = %c\n", *ptr);
    return 0;
}
```

char *ptr vs. char

ptr[]

```
char *ptr = "This is text";
```

```
char ptr[] = "This is text";
```

3)

- Αν χρησιμοποιήσουμε τον 2^ο τρόπο δήλωσης, δηλαδή:

```
char ptr[] = "This is text";
```

η μεταβλητή ptr είναι πίνακας, άρα το μέγεθος της μνήμης που δεσμεύεται για τη μεταβλητή ptr είναι ίσο με το πλήθος των χαρακτήρων του αλφαριθμητικού και μία επιπλέον θέση για τον τερματικό χαρακτήρα '\0'. Συγκεκριμένα, το μέγεθός της υπολογίζεται με την έκφραση `sizeof(ptr)`

- **Αντίθετα**, αν χρησιμοποιήσουμε τον 1^ο τρόπο δήλωσης, δηλαδή:

```
char *ptr = "This is text";
```

η μεταβλητή ptr είναι δείκτης σε χαρακτήρα, άρα το μέγεθος της μνήμης που δεσμεύεται για τη μεταβλητή ptr είναι ίσο με αυτό που δεσμεύεται για τον τύπο δείκτη (σε ένα 32-bit υπολογιστή το μέγεθός της είναι 4 bytes)

char *ptr vs. char ptr[]

```
char *ptr = "This is text";
```

```
char ptr[] = "This is text";
```

• 4)

- ▫ Αν χρησιμοποιήσουμε τον 1^ο τρόπο δήλωσης, δηλαδή:
 - `char *ptr = "This is text";`
 - η μνήμη που δεσμεύει ο μεταγλωττιστής για να αποθηκεύσει το
 - αλφαριθμητικό είναι συνήθως **μόνο για διάβασμα (read-only)** και δεν
 - επιτρέπεται να
 - Άρα, είναι πολύ

```
#include <stdio.h>
int main()
{
    char* ptr = "This is text";
    ptr[0] = 'a';
    return 0;
}
```

ς κατά την εκτέλεση του προγράμματος

- ▫ **Αντίθετα**, αν χρησιμοποιήσουμε τον 2^ο τρόπο δήλωσης, δηλαδή:

```
char ptr[] = "This is text";
```

επιτρέπεται να αλλάξουν τα περιεχόμενα του πίνακα, άρα εντολές όπως η
`ptr[0] = 'a';` εκτελούνται κανονικά

Διάβασμα Αλφαριθμητικών με την `scanf()`

- Για το διάβασμα ενός αλφαριθμητικού από το `stdin`, μπορούμε να χρησιμοποιήσουμε τη γνωστή μας συνάρτηση `scanf()`
- Για το διάβασμα αλφαριθμητικών με τη συνάρτηση `scanf()` χρησιμοποιείται το προσδιοριστικό `%s`
- Η συνάρτηση `scanf()` στην απλή χρήση της σταματάει το διάβασμα του αλφαριθμητικού, όταν συναντήσει τον κενό χαρακτήρα ή τον χαρακτήρα νέας γραμμής

```
#include <stdio.h>
int main()
{
    char str[100];

    printf("Enter text: ");
    scanf("%s", str);

    printf("Text: %s\n", str);
    return 0;
}
```

Ποια θα είναι η έξοδος του διπλανού προγράμματος, αν ο χρήστης πληκτρολογήσει το αλφαριθμητικό:
`We don't need no education...`

Έξοδος: `We`

Διάβασμα Αλφαριθμητικών με την `gets()`

- Για το διάβασμα ενός αλφαριθμητικού από το πληκτρολόγιο, μπορούμε να χρησιμοποιήσουμε τη συνάρτηση `gets()`
- Σε αντίθεση με τη συνάρτηση `scanf()`, η συνάρτηση `gets()` διαβάζει όλους τους χαρακτήρες μέχρι να συναντήσει τον χαρακτήρα νέας γραμμής (`'\n'`)
- Η `gets()` αντικαθιστά τον χαρακτήρα νέας γραμμής (`'\n'`) με τον τερματικό χαρακτήρα (`'\0'`)

```
#include <stdio.h>
int main()
{
    char str[100];

    printf("Enter text: ");
    gets(str);

    printf("Text: %s\n", str);
    return 0;
}
```

Ποια θα είναι η έξοδος του διπλανού προγράμματος, αν ο χρήστης πληκτρολογήσει το αλφαριθμητικό:
We don't need no education...

Έξοδος: We don't need no education...

Παρατηρήσεις (I)

- Και οι δύο συναρτήσεις (`scanf()` και `gets()`) δέχονται σαν παράμετρο έναν δείκτη προς χαρακτήρα
- Υπενθυμίζεται ότι το όνομα ενός πίνακα χωρίς αγκύλες είναι δείκτης στο πρώτο στοιχείο του πίνακα
- Επομένως, στα προηγούμενα παραδείγματα μπορεί να χρησιμοποιηθεί σαν παράμετρος το `str`, γιατί το `str` είναι δείκτης σε χαρακτήρα και συγκεκριμένα στον πρώτο χαρακτήρα του πίνακα
- Αρκετοί προγραμματιστές προτιμούν τη συνάρτηση `gets()` έναντι της συνάρτησης `scanf()`, γιατί διαβάζει και τους κενούς χαρακτήρες
- Προσοχή όμως, γιατί και οι δύο συναρτήσεις δεν είναι απολύτως ασφαλείς για το διάβασμα αλφαριθμητικών...(δείτε τη συνέχεια)



Παρατηρήσεις (II)



Ο δείκτης που δέχονται σαν παράμετρο οι συναρτήσεις `scanf()` και `gets()` πρέπει να δείχνει σε μία μνήμη που έχει δεσμευτεί για την αποθήκευση του αλφαριθμητικού που θα εισάγει ο χρήστης

Π.χ.

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

    printf("Enter text: ");
    gets(ptr);

    printf("Text: %s\n", ptr);
    return 0;
}
```

Για να μην υπάρχει πρόβλημα
π.χ.: `char ptr[100];`

Έξοδος: Δεν θα εκτελεστεί σωστά το πρόγραμμα, αφού δεν έχει δεσμευτεί μνήμη για την αποθήκευση του αλφαριθμητικού

Παρατηρήσεις (III)



Οι `scanf()` και `gets()` δεν ελέγχουν αν υπάρχει διαθέσιμος χώρος για την αποθήκευση όλων των χαρακτήρων του αλφαριθμητικού

- Επομένως, αν ο χρήστης εισάγει ένα αλφαριθμητικό που έχει περισσότερους χαρακτήρες από το μέγεθος της δεσμευμένης μνήμης, το πρόγραμμα θα έχει **απρόβλεπτη συμπεριφορά**



Μεγάλη προσοχή όταν χρησιμοποιείτε τις `scanf()` και `gets()` για να διαβάσετε αλφαριθμητικά. **Δεν είναι ασφαλείς...**

Παράδειγμα

- α Ποια είναι η έξοδος του παρακάτω προγράμματος αν ο χρήστης εισάγει το αλφαριθμητικό: You are a moonchild... ???

```
#include <stdio.h>
int main()
{
    int i;
    char str[5];

    i = 20;

    printf("Enter text: ");
    gets(str);

    printf("%s %d\n", str, i);
    return 0;
}
```

Ποιοι είπατε:

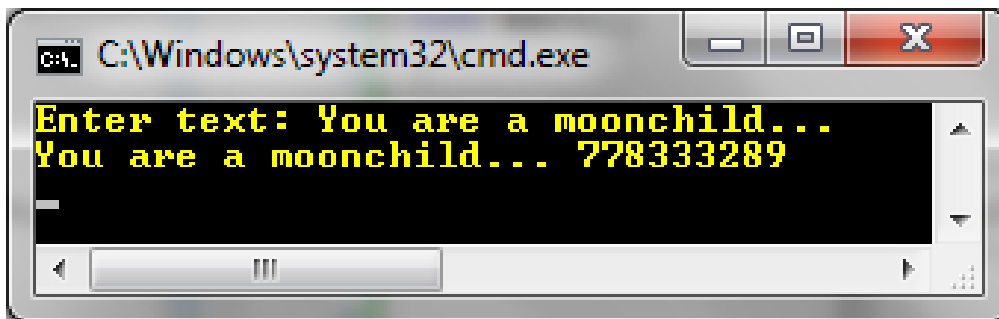
You a 20 ???

Και ποιοι λέτε:

You 20 ???

Είπε κανείς:

You are a moonchild... 20
(ίσως, με πιθανότητα 0.0001%)



Προσοχή στην υπέρβαση ορίων της επιτρεπόμενης δεσμευμένης μνήμης όταν χρησιμοποιείτε τις συναρτήσεις scanf() και gets()

Διάβασμα αλφαριθμητικών με μεγαλύτερη ασφάλεια (I)

- Όταν χρησιμοποιούμε τη συνάρτηση `gets()`, θεωρούμε, για λόγους απλότητας, ότι το μήκος του αλφαριθμητικού που διαβάζουμε από το πληκτρολόγιο είναι μέχρι ένα λογικό πλήθος χαρακτήρων (π.χ. 100)
- Ωστόσο, **αν θέλετε να αισθάνεστε πιο ασφαλείς** ότι δεν θα δημιουργηθεί πρόβλημα στις δικές σας εφαρμογές με το διάβασμα ενός αλφαριθμητικού από το πληκτρολόγιο, προτείνονται κι άλλες εναλλακτικές μέθοδοι, όπως:

1) Χρησιμοποιείτε τη συνάρτηση `fgets()`, η οποία είναι πιο ασφαλής από την `gets()` (περιγράφεται στο κεφάλαιο των αρχείων), γιατί καθορίζει το μέγιστο πλήθος των χαρακτήρων που θα αποθηκευτούν στον πίνακα, άρα αποφεύγεται η υπερχείλισή του

Για το διάβασμα ενός αλφαριθμητικού από το πληκτρολόγιο, μπορείτε απλώς να αντικαταστήσετε την:

```
gets(str)  
με την:  
fgets(str, sizeof(str), stdin)
```

(η παράμετρος `stdin` αναφέρεται στο πληκτρολόγιο)

Διάβασμα αλφαριθμητικών με μεγαλύτερη ασφάλεια (II)

- 2) Στη `scanf()` αντί για το προσδιοριστικό `%s` καλύτερα να χρησιμοποιείτε το `%ns`, όπου το `n` δηλώνει το μέγιστο πλήθος των χαρακτήρων που θα διαβαστούν
- 3) Το μέγεθος του πίνακα, στον οποίο θα αποθηκευτεί το αλφαριθμητικό, φροντίστε να είναι αρκετά μεγάλο, ώστε να «χωράει» σίγουρα το αλφαριθμητικό (π.χ. `char str[5000]`)
- 4) Μπορείτε χρησιμοποιώντας τη συνάρτηση `malloc()` να δεσμεύσετε δυναμικά πολλή μνήμη, η οποία να είστε σίγουροι ότι θα χωράει το αλφαριθμητικό (π.χ. 50.000 bytes), να αποθηκεύσετε το αλφαριθμητικό στη μνήμη αυτή και μετά, με χρήση της συνάρτησης `realloc()`, να μικραίνετε το μέγεθός της και να το κάνετε ίσο με το μήκος του αλφαριθμητικού

Περισσότερες λεπτομέρειες για τη `malloc()` και τη `realloc()` θα δείτε στη διάλεξη που αφορά τη διαχείριση μνήμης

Διάβασμα αλφαριθμητικών με μεγαλύτερη ασφάλεια (III)

5) Αν δεν σας καλύπτει τίποτα από τα προηγούμενα, τότε μπορείτε να... φτιάξετε έναν ατέρμονο `while` βρόχο, ο οποίος θα διαβάζει συνέχεια χαρακτήρες με τη συνάρτηση `getchar()` μέχρι είτε συναντηθεί ο χαρακτήρας `'\n'` είτε επιστραφεί η τιμή `EOF`

Οι χαρακτήρες θα αποθηκεύονται σε περιοχή της μνήμης που πρέπει να έχει δεσμευτεί δυναμικά με τη συνάρτηση `malloc()` (π.χ. αρχικό μέγεθος μνήμης 3000 bytes)

Όταν η δεσμευμένη αυτή μνήμη γεμίσει, τότε πρέπει να καλείτε τη συνάρτηση `realloc()` για να μεγαλώσετε το μέγεθός της (π.χ. την πρώτη φορά που θα γεμίσει το νέο μέγεθός της να γίνει 6000 bytes, την επόμενη 9000 bytes, κ.ο.κ, δηλαδή κάθε φορά που γεμίζει να προσθέτετε 3000 bytes)

Φυσικά, όλοι οι παραπάνω τρόποι είναι σίγουρα **πολύ ασφαλείς**, αλλά ταυτόχρονα και «**παρανοϊκοί**» (ή τουλάχιστον **περιττοί**, για ένα μη-επαγγελματικό πρόγραμμα ενός πρωτοετούς φοιτητή...), δηλ. στο πλαίσιο του μαθήματος, δεν χρειάζεται να τους ακολουθήσετε...

Παραδείγματα (I)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει ένα αλφαριθμητικό (μέχρι 100 χαρακτήρες) και να εμφανίζει τον αριθμό των χαρακτήρων του χωρίς τη χρήση της συνάρτησης `strlen()`

```
#include <stdio.h>
int main()
{
    int i;
    char str[100];

    printf("Enter text: ");
    gets(str);

    for(i = 0; str[i] != '\0'; i++)
        ;

    printf("Text Length = %d\n", i);
    return 0;
}
```

Παραδείγματα (II)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
    char* ptr;
    char str[] = "another";

    ptr = str;
    printf("%d %c\n", *ptr + 2, *(ptr + 2));
    return 0;
}
```

Έξοδος: 99 o

Παραδείγματα (III)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει 10 φορές ένα αλφαριθμητικό (μέχρι 100 χαρακτήρες) και να εμφανίζει το πλήθος των πεζών και κεφαλαίων γραμμάτων που περιέχονται σε αυτό

```
#include <stdio.h>
int main()
{
    int i,j,small_let,big_let;
    char str[100];

    for(i = 0; i < 10; i++)
    {
        printf("Enter text: ");
        gets(str);

        small_let = big_let = 0;
        for(j = 0; str[j] != '\0'; j++)
        {
            if(str[j] >= 'a' && str[j] <= 'z')
                small_let++;
            else if(str[j] >= 'A' && str[j] <= 'Z')
                big_let++;
        }
        printf("%d small and %d big\n",small_let,big_let);
    }
    return 0;
}
```


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'`)
- Το όνομά της προκύπτει από την έκφραση «string length»
- Η συνάρτηση `strlen()` δέχεται σαν παράμετρο έναν δείκτη προς τη μνήμη που είναι αποθηκευμένη το αλφαριθμητικό
- Το πρωτότυπό της δηλώνεται ως εξής:

```
size_t strlen(const char *str);
```
- Ο δείκτης δηλώνεται ως `const` ώστε να μην μπορεί η `strlen()` να μεταβάλει το περιεχόμενο του αλφαριθμητικού
- Ο τύπος `size_t` είναι δηλωμένος στην C βιβλιοθήκη σαν απρόσημος ακέραιος (συνήθως ως `unsigned int`)

Παραδείγματα (I)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει ένα αλφαριθμητικό μέχρι 100 χαρακτήρες και να εμφανίζει το πλήθος των χαρακτήρων του με χρήση της συνάρτησης `strlen()`

```
#include <stdio.h>
#include <string.h>
int main()
{
    int len;
    char str[100];

    printf("Enter text: ");
    gets(str);

    len = strlen(str);
    printf("Text has %d characters\n",len); /* Θα μπορούσαμε
να μην χρησιμοποιήσουμε τη μεταβλητή len και να γράψουμε:
printf("Text has %d characters\n",strlen(text)); */
    return 0;
}
```

Παραδείγματα (II)

- Γράψτε ένα πρόγραμμα το οποίο να διαβάζει ένα αλφαριθμητικό μέχρι 100 χαρακτήρες και να το εμφανίζει μόνο εάν οι δύο τελευταίοι χαρακτήρες του είναι ίσοι με 'a'

```
#include <stdio.h>
#include <string.h>
int main()
{
    int len;
    char str[100];

    printf("Enter text: ");
    gets(str);

    len = strlen(str);

    if(len > 1 && str[len-1] == 'a' && str[len-2] == 'a')
        printf("Text: %s\n",str);
    return 0;
}
```

Η συνάρτηση strcpy()

- Η συνάρτηση `strcpy()` 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 πρωτότυπο 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`)
- Π.χ. η παρακάτω εντολή αντιγράφει το αλφαριθμητικό "something" στον πίνακα `str`

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

Παρατηρήσεις



Η `strcpy()` δεν ελέγχει αν η μνήμη - στην οποία θα αντιγραφεί το αλφαριθμητικό – χωράει όλους τους χαρακτήρες του, οπότε 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. το επόμενο πρόγραμμα δεν θα εκτελεστεί σωστά (θα εμφανιστεί λάθος κατά την εκτέλεση του προγράμματος), γιατί το μέγεθος του πίνακα `str` είναι μικρότερο από ότι θα έπρεπε

```
char str[5];  
strcpy(str, "something");
```



Πριν την αντιγραφή του αλφαριθμητικού **πρέπει να έχει προηγηθεί** η δέσμευση της αντίστοιχης μνήμης

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()
{
    char str1[100],str2[100];

    printf("Enter text: ");
    gets(str2);

    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()
{
    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`

Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

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

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

    strncpy(str1, str3, 5);
    printf("%s\n", str1);

    return 0;
}
```

Έξοδος: New test
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

- Να προσέχετε, ώστε το μέγεθος της μνήμης που έχει δεσμευτεί για το 1ο αλφαριθμητικό να είναι αρκετά μεγάλο για να χωράει το άθροισμα των χαρακτήρων του 1ου και 2ου αλφαριθμητικού
- Αν δεν είναι, τότε οι πλεονάζοντες χαρακτήρες θα εγγραφούν σε μνήμη μετά από το επιτρεπτό όριο, υπερεγγράφοντας τα δεδομένα που υπάρχουν εκεί
- Ισχύει επίσης 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 <string.h>
int main()
{
    char str1[100],str2[100];
    char str3[200] = {0}; /* Ο πίνακας str3 ορίζεται ως
    πίνακας 200 χαρακτήρων, γιατί σε αυτόν θα αποθηκευτούν όλοι οι
    χαρακτήρες των πινάκων str1 και str2. Άρα, το μέγιστο πλήθος
    χαρακτήρων που μπορεί να αποθηκευτεί στον str3 πίνακα είναι
    100+100=200 */

    printf("Enter first text: ");
    gets(str1);

    printf("Enter second text: ");
    gets(str2);

    strcat(str3,str1); /* Ένωση των περιεχομένων των str3 και
    str1 πινάκων και αποθήκευση του αποτελέσματος στον str3 πίνακα.
    Ο πίνακας str3 περιέχει μόνο τον τερματικό χαρακτήρα, άρα στον
    πίνακα str3 θα αποθηκευτεί ο πίνακας str1. */

    strcat(str3,str2); /* Ένωση των περιεχομένων των str3 και
    str2 πινάκων και αποθήκευση του αποτελέσματος στον str3 πίνακα.
    Από προηγουμένως, ο πίνακας str3 περιέχει τους χαρακτήρες του
    str1 πίνακα, άρα στον πίνακα str3 θα προστεθούν και τα
    περιεχόμενα του πίνακα str2. */

    printf("The merged text is %s\n",str3);
    return 0;
}
```

Η συνάρτηση `strcmp()`

- 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()`

- Ένα αλφαριθμητικό **θεωρείται μικρότερο** από κάποιο άλλο αν:
 - α) η **τιμή του πρώτου μη κοινού χαρακτήρα είναι μικρότερη** από την τιμή του αντίστοιχου χαρακτήρα στο δεύτερο αλφαριθμητικό
 - β) οι **χαρακτήρες** τους είναι οι **ίδιοι**, αλλά το **μήκος** του 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("onE", "one");
```

επιστρέφει μία αρνητική τιμή,

ενώ η εντολή:

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

επιστρέφει μία θετική τιμή, αντίστοιχα

Η συνάρτηση `strcmp()`

- 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

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

- Γράψτε ένα πρόγραμμα το οποίο 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()`.

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

    printf("Enter first text: ");
    gets(str1);

    printf("Enter second text: ");
    gets(str2);

    ret = strcmp(str1,str2);

    /* Θα μπορούσαμε να μη χρησιμοποιήσουμε τη μεταβλητή ret
    και να γράψουμε if(strcmp(str1,str2) == 0) */
    if(ret == 0)
        printf("The texts are the same\n");
    else
    {
        printf("The texts are different\n");
        if(strncmp(str1,str2,3) == 0)
            printf("The first 3 chars are the same\n");
    }
    return 0;
}
```


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

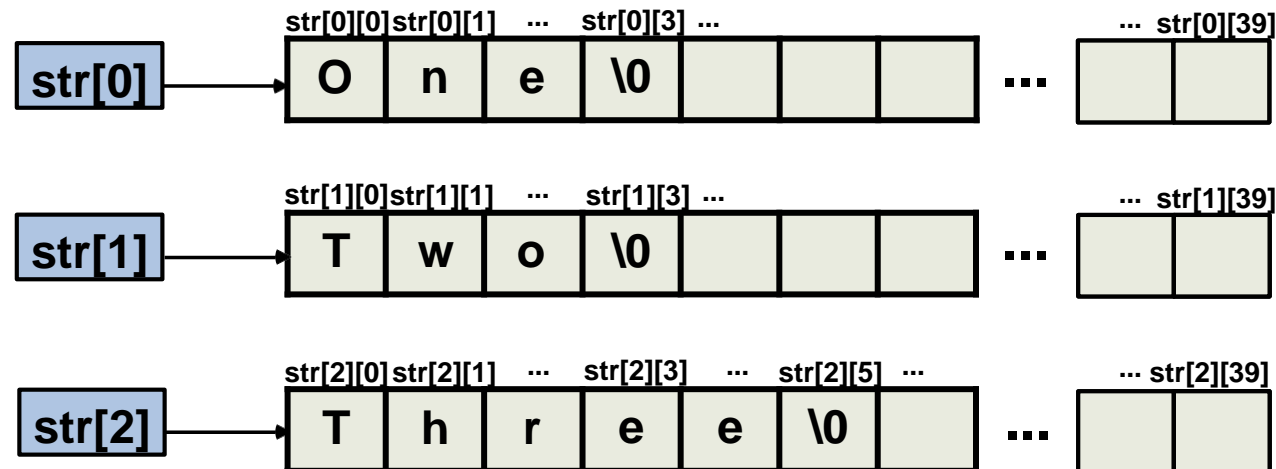
Π.χ. με τη δήλωση:

```
char str[3][40] = {"One", "Two", "Three"};
```

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

Διδιάστατοι πίνακες και αλφαριθμητικά (II)

- Θυμηθείτε από την ενότητα των «Δεικτών» ότι μπορούμε να χειριστούμε το καθένα από τα N στοιχεία $str[0], str[1], \dots, str[N-1]$ ενός διδιάστατου πίνακα, έστω $str[N][M]$, σαν δείκτη σε πίνακα που περιέχει τα M στοιχεία της αντίστοιχης γραμμής.
- Άρα, στο προηγούμενο παράδειγμα, το $str[0]$ μπορεί να χρησιμοποιηθεί σαν δείκτης σε έναν πίνακα 40 χαρακτήρων, ο οποίος περιέχει το αλφαριθμητικό "One", ενώ με παρόμοιο τρόπο μπορούμε να χειριστούμε και τα στοιχεία $str[1]$ και $str[2]$



Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main()
{
    char arr[7][10] = {"Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday", "Sunday"};
    int i;

    for(i = 0; i < 7; i++)
        if(arr[i][2] == 'n' && arr[i][3] == 'd' &&
*(arr[i]+4) == 'a')
            printf("%s is No.%d week day\n", arr[i], i+1);

    return 0;
}
```

Έξοδος:

Monday is No.1 week day

Sunday is No.7 week day