Προγραμματισμός Ι

Αλφαριθμητικά

Πανεπιστήμιο Πελοποννήσου
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Νικόλαος Δ. Τσελίκας

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 Ν χαρακτήρες** θα **πρέπει να δηλώσουμε** έναν πίνακα με **Ν+1** θέσεις χαρακτήρων

Αποθήκευση Αλφαριθμητικών με τη Δήλωση (Ι)

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 (ΙΙ)

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 (ΙΙΙ)

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 (Ι)

- 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 (ΙΙ)

- 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 (ΙΙΙ)

- 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 (Ι)

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

```
#include <stdio.h>
int main(void)
{
        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 (ΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

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

Έξοδος: 17

Παραδείγματα (ΙΙΙ)

- 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(void)
{
        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() (Ι)

- 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'`)

Σε περίπτωση που δεν βρεθεί ο τερματικός χαρακτήρας, η `printf()` συνεχίζει να προσπελαύνει τη μνήμη και μετά το τέλος της μνήμης που έχει δεσμευτεί για το αλφαριθμητικό εμφανίζοντας χαρακτήρες, έως ότου συναντήσει τον τερματικό χαρακτήρα...!!!

- Στο διπλανό παράδειγμα χρησιμοποιείται το όνομα του πίνακα (αφού το όνομα ενός πίνακα είναι δείκτης στο πρώτο του στοιχείο)

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

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

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

- 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(void)
{
        char str[] = "This is text";
        printf("%s\n", str+5);
        return 0;
}
```

ή ισοδύναμα:

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

Εμφάνιση Αλφαριθμητικών με την printf() (III)

- Αν η printf() συναντήσει τον τερματικό χαρακτήρα μέσα σε ένα αλφαριθμητικό, σταματάει την εμφάνιση των υπολοίπων χαρακτήρων, π.χ.:

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

        str[4] = '\0';
        printf("%s\n", str);
        return 0;
}
```

- Αφού βρέθηκε ο τερματικός χαρακτήρας στην 5η θέση, η printf() εμφανίζει Samp και αγνοεί τους υπόλοιπους χαρακτήρες του αλφαριθμητικού str

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(void)
{
        char str[] = "This is text";

        puts(str);
        return 0;
}
```

Παράδειγμα (I)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```c
#include <stdio.h>
int main(void)
{
        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╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟
╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟
╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟╟Ξ[|9Tn▼
```

Παράδειγμα (ΙΙ)

- Ποια είναι η έξοδος των παρακάτω προγραμμάτων ???

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

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

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

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

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

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

Έξοδος: **ab**

Παράδειγμα (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
        char str[] = "Right'0'Wrong";

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

Απαντήσατε `Right` ??? Ή μήπως `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 οι απαντήσεις σας σωστές, θα έπρεπε να είχαμε δηλώσει το αλφαριθμητικό ως: `char str[] = "Right\0Wrong";`

ή `char str[] = "Right'\0'Wrong";` αντίστοιχα.

Δείκτες και Κυριολεκτικά Αλφαριθμητικά (Ι)

- Αφού ένα κυριολεκτικό αλφαριθμητικό αποθηκεύεται σαν πίνακας χαρακτήρων, μπορούμε να το χρησιμοποιήσουμε και σαν δείκτη με τύπο `char*`

- Με το επόμενο πρόγραμμα βλέπουμε αυτή τη σχέση με έναν πραγματικά ιδιόμορφο τρόπο και, όσο «παρανοϊκό» κι αν φαίνεται, το πρόγραμμα εμφανίζει τον πέμπτο χαρακτήρα του κυριολεκτικού αλφαριθμητικού `"message"`, δηλαδή το `'a'`, με χρήση του ίδιου του κυριολεκτικού αλφαριθμητικού σαν πίνακα και σαν δείκτη

```
#include <stdio.h>
int main(void)
{
        printf("%c %c\n", "message"[4], *("message"+4));
        return 0;
}
```

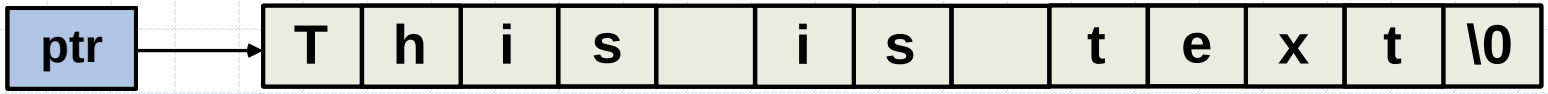
Δείκτες και Κυριολεκτικά Αλφαριθμητικά (ΙΙ)

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

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

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

- Με την εντολή `ptr = "This is text";` ο μεταγλωττιστής δεσμεύει μνήμη για να αποθηκεύσει τους χαρακτήρες του κυριολεκτικού αλφαριθμητικού και τον τερματικό χαρακτήρα ενώ, στη συνέχεια, ο δείκτης `ptr` δείχνει σε αυτή τη μνήμη και συγκεκριμένα στη διεύθυνση του πρώτου χαρακτήρα, όπως φαίνεται στο ακόλουθο σχήμα:



Δείκτες και Κυριολεκτικά Αλφαριθμητικά (ΙΙΙ)

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

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

και

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

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

- Και γιατί αυτό????

Θυμηθείτε ότι το όνομα ενός πίνακα (χωρίς τις αγκύλες) είναι <u>δείκτης στο πρώτο του στοιχείο</u>

Δείκτες και Κυριολεκτικά Αλφαριθμητικά (IV)

- Τους χαρακτήρες ενός κυριολεκτικού αλφαριθμητικού μπορούμε να τους χειριστούμε χρησιμοποιώντας είτε σημειογραφία δείκτη είτε σημειογραφία πίνακα

- Το παρακάτω παράδειγμα χρησιμοποιεί και τους δύο τρόπους για να εμφανίσει «χαρακτήρα-χαρακτήρα» όλους τους χαρακτήρες του κυριολεκτικού αλφαριθμητικού

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

        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'`

Παρατηρήσεις

Επειδή η μνήμη που δεσμεύει ο μεταγλωττιστής για να αποθηκεύσει ένα κυριολεκτικό αλφαριθμητικό είναι (συνήθως) μόνο για ανάγνωση (read only memory) δεν επιτρέπεται (συνήθως) η τροποποίηση ενός κυριολεκτικού αλφαριθμητικού

- Για παράδειγμα, στο επόμενο πρόγραμμα θα εμφανιστεί μήνυμα λάθους κατά την εκτέλεσή του:

```
#include <stdio.h>
int main(void)
{
        char *ptr = "This is text";

        ptr[0] = 'a';
        return 0;
}
```

char ptr[] vs. char *ptr

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

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

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

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

μετά τη δήλωση και την αρχικοποίηση αυτή, **επιτρέπεται να αποθηκεύσουμε** στον πίνακα `ptr` ένα άλλο κυριολεκτικό 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'`)

Επιτρέπεται π.χ. να γράψουμε `ptr[0] = 'a';` χωρίς καμία ανησυχία

char ptr[] vs. char *ptr

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

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

2) **Αντίθετα**, αν έχουμε χρησιμοποιήσει τον 2ο 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 κυριολεκτικό αλφαριθμητικό, **ακόμα και αν αυτό έχει περισσότερους χαρακτήρες**

ΠΡΟΣΟΧΗ! Μην ξεχνάμε βέβαια ότι μία εντολή όπως π.χ.

`ptr[0] = 'a';` πιθανότατα θα προκαλέσει λάθος κατά την εκτέλεση του προγράμματος (runtime error)

Παράδειγμα

```
#include <stdio.h>
int main(void)
{
        char *ptr = "First text";

        ptr = "This is a new text";
        printf("%c\n", *ptr);
        return 0;
}
```

- Ο δείκτης **ptr** δείχνει αρχικά στον πρώτο χαρακτήρα του κυριολεκτικού αλφαριθμητικού **"First text"**

- Στη συνέχεια, η εντολή: **ptr = "This is a new text";** κάνει τον δείκτη **ptr** να δείχνει σε νέα μνήμη που έχει δεσμευτεί για να αποθηκεύσει το δεύτερο κυριολεκτικό αλφαριθμητικό

- Δεδομένου ότι ο δείκτης **ptr** δείχνει στον πρώτο χαρακτήρα του κυριολεκτικού αλφαριθμητικού, το ***ptr** είναι ίσο με **'T'** και το πρόγραμμα εμφανίζει : **T**

Παρατηρήσεις

Πριν χρησιμοποιήσουμε έναν δείκτη θα πρέπει να τον έχουμε κάνει να δείχνει προς μία έγκυρη διεύθυνση μνήμης, π.χ.:

Είναι σωστές οι αναθέσεις αυτές???

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

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

Φυσικά και ΟΧΙ!!! Αφού ο `ptr` δεν έχει αρχικοποιηθεί, η εγγραφή των χαρακτήρων `'a'`, `'b'` και `'\0'` κάπου στη μνήμη μπορεί να προκαλέσει απρόβλεπτη συμπεριφορά στο πρόγραμμα.

Στο διά ταύτα, μην ξεχνάτε ότι η χρήση ενός μη-αρχικοποιημένου δείκτη είναι πολύ σοβαρό λάθος

Παράδειγμα

- Διορθώστε το προηγούμενο πρόγραμμα, ώστε να δουλεύει σωστά...

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

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

```
#include <stdio.h>
int main(void)
{
        char str[3], *ptr;
        ptr = str;

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

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(void)
{
        char str[100];
        int i = 10;

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

Ποια θα είναι η έξοδος του διπλανού προγράμματος, αν ο χρήστης πληκτρολογήσει το αλφαριθμητικό:

We don't need no education...

Έξοδος: We 10

Παρατηρήσεις (1/4)

- Για να υποχρεώσουμε τη `scanf()` να διαβάσει όλους τους χαρακτήρες πρέπει να χρησιμοποιήσουμε μία πιο σύνθετη μορφή της, όπως:

```
scanf("%[^\n]",str);
```

Παρατηρήσεις (2/4)

Η `scanf()` δεν ελέγχει αν υπάρχει διαθέσιμη μνήμη για την αποθήκευση όλων των χαρακτήρων, επομένως, αν ο χρήστης εισάγει περισσότερους χαρακτήρες απ' ότι η δεσμευμένη μνήμη, το πρόγραμμα θα έχει απρόβλεπτη συμπεριφορά

- Π.χ., θεωρήστε ότι στο διπλανό πρόγραμμα ο χρήστης εισάγει ένα αλφαριθμητικό με περισσότερους από 20 χαρακτήρες

```
#include <stdio.h>
int main(void)
{
        char str[20];
        int i = 10;

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

- Οι πρώτοι 20 χαρακτήρες θα αποθηκευτούν στο `str`, αλλά οι υπόλοιποι θα αποθηκευτούν σε μνήμη εκτός των ορίων του πίνακα, γεγονός που προκαλεί την απρόβλεπτη συμπεριφορά του προγράμματος
- Π.χ., αν η μεταβλητή `i` βρίσκεται σε εκείνο το κομμάτι της μνήμης, η τιμή της θα αλλάξει

Παρατηρήσεις (3/4)

- Για την αποφυγή της υπερχείλισης μνήμης, μπορούμε να χρησιμοποιήσουμε `%ns` αντί για `%s`, όπου το `n` καθορίζει τον μέγιστο αριθμό των χαρακτήρων που θα αποθηκευτούν
- Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε τη συνάρτηση `fgets()` (θα δούμε την `fgets()` στο Κεφ. 15)
- Μία άλλη συνάρτηση για διάβασμα αλφαριθμητικών είναι και η `gets()`, που δηλώνεται στο `stdio.h`:

```
char *gets(char *str);
```

- Όπως και η `scanf()`, έτσι κι η `gets()` διαβάζει χαρακτήρες απ' το `stdin` και τους αποθηκεύει στη μνήμη που δείχνει ο δείκτης `str`
- Δεδομένου ότι δεν είναι δυνατόν να καθορίσουμε τον μέγιστο αριθμό των χαρακτήρων που θα διαβαστούν, η `gets()` είναι άκρως επικίνδυνη και υπάρχει πάντα το ενδεχόμενο υπερχείλισης μνήμης

Μην χρησιμοποιείτε τη `gets()` - Δεν είναι καθόλου ασφαλής!

Η `fgets()` **είναι ασφαλέστερη απ' την `gets()`, διότι μπορεί να καθοριστεί ο μέγιστος αριθμός χαρακτήρων που θα διαβαστούν**

Παρατηρήσεις (4/4)

Ο δείκτης που μεταβιβάζεται στη συνάρτηση διαβάσματος **πρέπει να δείχνει σε μνήμη που έχει ήδη δεσμευτεί** για την αποθήκευση του αλφαριθμητικού

- Π.χ.:

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

        printf("Enter text: ");
        fgets(ptr, sizeof(ptr), stdin);
        printf("%s\n", ptr);
        return 0;
}
```

- Αφού ο δείκτης `ptr` δεν δείχνει σε δεσμευμένη μνήμη, το πρόγραμμα δεν θα εκτελεστεί σωστά

- Αν αντί για `char *ptr;` γράψουμε `char ptr[100];` το πρόγραμμα θα εκτελεστεί σωστά, γιατί τώρα δεσμεύτηκε μνήμη για την αποθήκευση του αλφαριθμητικού

Ασφαλές διάβασμα αλφαριθμητικών σε συστήματα περιορισμένης μνήμης

- Αν η εφαρμογή που υλοποιείτε πρόκειται να εκτελεστεί σε ένα σύστημα με περιορισμένη μνήμη και θέλετε η μνήμη που θα δεσμευτεί να μην είναι μεγαλύτερη από όση χρειάζεται, δημιουργήστε έναν βρόχο και χρησιμοποιήστε την `getchar()` για να διαβάζετε τους χαρακτήρες, μέχρι να συναντηθεί ο χαρακτήρας `'\n'` ή να επιστραφεί η τιμή `EOF`

- Όπως θα δούμε στο Κεφ.14, οι χαρακτήρες θα πρέπει να αποθηκεύονται σε μία δυναμικά δεσμευμένη μνήμη (π.χ. αρχικό μέγεθος 500 bytes)

- Κάθε φορά που η μνήμη γεμίζει, χρησιμοποιήστε τη `realloc()` για να αυξήσετε το μέγεθός της, προσθέτοντας το αρχικό της μέγεθος, π.χ., την πρώτη φορά που θα γεμίσει, προσθέστε 500 για να γίνει το νέο μέγεθός της 1000 byte, τη δεύτερη φορά προσθέστε άλλες 500 για να γίνει το μέγεθός της 1500, κ.ο.κ.

- Όταν όλοι οι χαρακτήρες έχουν διαβαστεί καλέστε για μία τελευταία φορά τη `realloc()`, ώστε να κάνετε το μέγεθος της δεσμευμένης μνήμης ίσο με το μήκος του αλφαριθμητικού

Διάβασμα αλφαριθμητικών – 1η πρόταση

- Δεδομένων των προβλημάτων που παρουσιάστηκαν για τις `scanf()` και `gets()`, χρησιμοποιείστε τη συνάρτηση `fgets()`, η οποία είναι ασφαλέστερη (περιγράφεται στο Κεφ. 15), γιατί καθορίζει το μέγιστο πλήθος των χαρακτήρων που θα αποθηκευτούν στον πίνακα χαρακτήρων, άρα αποφεύγεται η υπερχείλισή του

- Μπορείτε δηλαδή για το διάβασμα ενός αλφαριθμητικού από το πληκτρολόγιο, να χρησιμοποιείτε για απλότητα την `fgets()` για να διαβάζουμε χαρακτήρες, θεωρώντας ότι ο χρήστης δεν θα εισάγει περισσότερους χαρακτήρες από ένα λογικό όριο (π.χ. 100 χαρακτήρες)

```
fgets(str, sizeof(str), stdin)
```

(η παράμετρος `stdin` αναφέρεται στο πληκτρολόγιο)

Διάβασμα αλφαριθμητικών – 2η πρόταση (1/2)

- Επίσης, μπορείτε να χρησιμοποιείτε τη συνάρτηση **read_text()** (που κάνει χρήση της **fgets()**) και έχουμε ήδη υλοποιήσει

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

	if(fgets(str, size, stdin) == NULL)
	{
		printf("Error: fgets() failed\n");
		exit(EXIT_FAILURE);
	}
	len = strlen(str);
	if(len > 0)
	{
		if(flag && (str[len-1] == '\n'))
		{
			str[len-1] = '\0';
			len--;
		}
	}
	else
	{
		printf("Error: No input\n");
		exit(EXIT_FAILURE);
	}
	return len;
}
```

Διάβασμα αλφαριθμητικών – 2η πρόταση (2/2)

- Η `read_text()` χρησιμοποιεί την `fgets()` για να διαβάσει χαρακτήρες απ' το `stdin`, το οποίο εξ' ορισμού συνδέεται με το πληκτρολόγιο
- Τους αποθηκεύει στον πίνακα `str` μήκους `size` bytes
- Δεδομένου ότι το `'\n'` μπορεί επίσης να αποθηκευτεί, η `read_text()` δίνει τη δυνατότητα να το απομακρύνουμε περνώντας οποιαδήποτε μη μηδενική τιμή ως `flag`
- Η `read_text()` χρησιμοποιεί τη `strlen()` για να επιστρέψει το μήκος του αλφαριθμητικού
- Αν κάτι δεν πάει καλά, καλείται η συνάρτηση `exit()`, που δηλώνεται στο `stdlib.h` και τερματίζει άμεσα το πρόγραμμα
- Η `exit()` δέχεται ως παράμετρο μία ακέραια τιμή, που δηλώνει την κατάσταση τερματισμού
  - Η τιμή `0` δηλώνει τον κανονικό τερματισμό του προγράμματος
  - Επίσης, η C παρέχει τις `EXIT_SUCCESS` και `EXIT_FAILURE` που δηλώνονται σαν μακροεντολές στο `stdlib.h` (με τυπικές τιμές `0` και `1`)
  - Η κύρια διαφορά της `exit()` με την εντολή `return` είναι όταν καλείται μέσα από μία συνάρτηση, εκτός της `main()`, αφού η `return` προκαλεί τον τερματισμό της συνάρτησης, ενώ η `exit()` τον τερματισμό του προγράμματος

Παραδείγματα (Ι)

- Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει ένα αλφαριθμητικό μέχρι 100 χαρακτήρες και να εμφανίζει το πλήθος των χαρακτήρων του, τον αριθμό των εμφανίσεων του χαρακτήρα `'b'`, καθώς και το αλφαριθμητικό, αφού πρώτα αντικαταστήσει κάθε κενό χαρακτήρα με τον χαρακτήρα νέας γραμμής και τα `'a'` με `'p'`. (μην χρησιμοποιήσετε την τιμή επιστροφής της `read_text()` για το πλήθος των χαρακτήρων του)

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

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

int main(void)
{
	char str[100];
	int i, cnt;

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

	cnt = 0;
	for(i = 0; str[i] != '\0'; i++)
	{
		if(str[i] == ' ')
			str[i] = '\n';
		else if(str[i] == 'a')
			str[i] = 'p';
		else if(str[i] == 'b')
			cnt++;
	}
	printf("Len = %d Times = %d\nText = %s\n", i, cnt, str);
	return 0;
}
```

Παραδείγματα (ΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

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

        for(; *str; printf("%s ", str++));
        return 0;
}
```

Έξοδος: `this his is s`

Παραδείγματα (ΙΙΙ)

- Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει ένα αλφαριθμητικό μέχρι 100 χαρακτήρες και να ελέγχει αν είναι «παλίνδρομο», δηλαδή αν μπορεί να διαβαστεί με τον ίδιο τρόπο και από το τέλος προς την αρχή (π.χ. η λέξη **`level`** είναι παλίνδρομο, γιατί διαβάζεται με τον ίδιο τρόπο και από τις δύο κατευθύνσεις).

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

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

int main(void)
{
	char str[100];
	int i, diff, len;

	printf("Enter text: ");
	len = read_text(str, sizeof(str), 1);

	diff = 0;
	for(i = 0; i < len/2; i++)
	{
		if(str[i] != str[len-1-i]) /* If two characters are not
the same, the loop terminates. */
		{
			diff = 1;
			break;
		}
	}
	if(diff == 1)
		printf("%s is not a palindrome\n", str);
	else
		printf("%s is a palindrome\n", str);
	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 (Ι)

- Π.χ.:

```
int len;
len = strlen("Test"); /* Το len είναι 4. */

char *p = "";
len = strlen(p); /* Το len είναι 0. */

char str[] = "This is a test";
len = strlen(str); /* Το len είναι 14. */
```

Παραδείγματα (ΙΙ)

- Να γραφεί ένα πρόγραμμα το οποίο να διαβάζει ένα αλφαριθμητικό μέχρι 100 χαρακτήρες και να το εμφανίζει, αφού πρώτα αντικαταστήσει όλους τους 'a' χαρακτήρες που βρίσκονται στην αρχή ή/και στο τέλος του αλφαριθμητικού με τον κενό χαρακτήρα. Για παράδειγμα, αν ο χρήστης εισάγει το αλφαριθμητικό "aabcdaa" το πρόγραμμα να εμφανίζει " bcd ", ενώ αν εισάγει το "bcdaa" να εμφανίζει "bcd "

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

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

int main(void)
{
	char str[100];
	int i, len;

	printf("Enter text: ");
	len = read_text(str, sizeof(str), 1);

	for(i = 0; i < len; i++)
	{
		if(str[i] == 'a')
			str[i] = ' ';
		else
			break;
	}
	for(i = len-1; i >= 0; i--)
	{
		if(str[i] == 'a')
			str[i] = ' ';
		else
			break;
	}
	printf("Text: %s\n", str);
	return 0;
}
```

Παραδείγματα (ΙΙΙ)

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

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

        printf("%d %d\n", strlen(str+4), strlen("Text"+1));
        return 0;
}
```

Έξοδος: 0 3

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 (Ι)

- 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 (ΙΙ)

- 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

```c
#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 πρώτου αλφαριθμητικού 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()` (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

```c
#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 (Ι)

- 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:

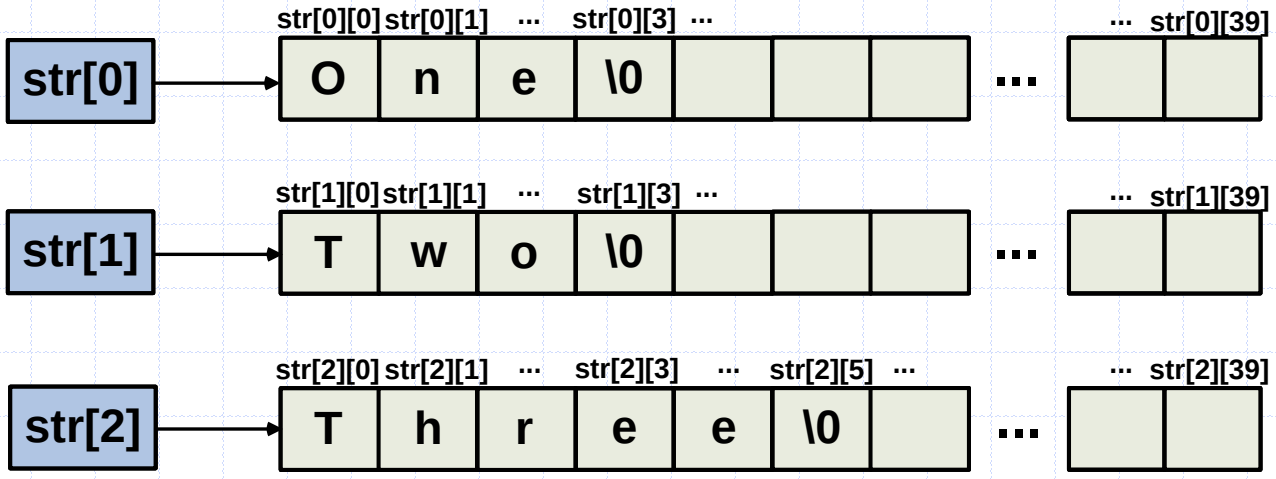
```
char str[3][40] = {"One", "Two", "Three"};
```

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

Διδιάστατοι πίνακες και αλφαριθμητικά (ΙΙ)

- Θυμηθείτε από την ενότητα των «Δεικτών» ότι μπορούμε να χειριστούμε το καθένα από τα `N` στοιχεία `str[0]`, `str[1]`, ..., `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]`



Διδιάστατοι πίνακες και αλφαριθμητικά (ΙΙΙ)

- Θυμηθείτε επίσης από την ενότητα «Πίνακας Δεικτών» ότι για την αποθήκευση αλφαριθμητικών, αντί για διδιάστατο πίνακα, για αποφυγή σπατάλης μνήμης μπορούμε να χρησιμοποιήσουμε έναν πίνακα δεικτών:

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

και για το κάθε αλφαριθμητικό δεσμεύεται όση μνήμη ακριβώς χρειάζεται, ενώ, όπως γνωρίζουμε, τον πίνακα μπορούμε να τον διαχειριστούμε σαν διδιάστατο για να έχουμε πρόσβαση σε κάθε χαρακτήρα των αλφαριθμητικών, π.χ.:

```
for(i = 0; i < 3; i++)
    if(str[i][0] == 'T')
        printf("%s\n", str[i]);
```

- Ο παραπάνω βρόχος εμφανίζει τα αλφαριθμητικά που αρχίζουν από **T**

Παράδειγμα

- Ποια είναι η έξοδος του παρακάτω προγράμματος ???

```
#include <stdio.h>
int main(void)
{
	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
```