Εργαστήριο 8: Συμβολοσειρές και Ορίσματα Γραμμής Εντολών

Στο εργαστήριο αυτό θα δούμε πώς ορίζονται και πώς χρησιμοποιούνται οι συμβολοσειρές (strings) στην C. Επίσης, θα μελετήσουμε κάποιες από τις συναρτήσεις της πρότυπης βιβλιοθήκης της C, που διευκολύνουν την επεξεργασία των συμβολοσειρών, τα πρωτότυπα των οποίων ορίζονται στο αρχείο επικεφαλίδας `string.h`. Τέλος, θα δούμε πώς να διαχειριζόμαστε στο πρόγραμμά μας τα ορίσματα που δίνονται στη γραμμή εντολής κατά την εκτέλεση ενός προγράμματος (command line arguments).

Άσκηση 1: Επεξεργασία συμβολοσειρών - (string.c)

1.1 Υλοποιήστε τη συνάρτηση `int mystrlen(char *str)` η οποία δέχεται σαν όρισμα μία συμβολοσειρά και επιστρέφει το μήκος της (χωρίς να συμπεριλαμβάνεται ο χαρακτήρας τέλους συμβολοσειράς `'\0'`).

1.2 Υλοποιήστε τη συνάρτηση `char *mystrcat(char *s1, char *s2)` η οποία προσαρτά ένα αντίγραφο της συμβολοσειράς `s2` στο τέλος της `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

1.3 Κατασκευάστε το πρόγραμμα `string.c`, το οποίο θα συμπεριλαμβάνει το αρχείο επικεφαλίδας `string.h` και θα πραγματοποιεί το σενάριο που ακολουθεί. Επίσης, ενσωματώστε στο πρόγραμμα και τις συναρτήσεις που υλοποιήσατε στην προηγούμενη άσκηση.

1.3.1 Ορίστε τις συμβολοσειρές `strA` και `strB` με στατική ή δυναμική δέσμευση μνήμης 80 χαρακτήρων.

1.3.2 Αντιγράψτε στην `strA` τη συμβολοσειρά `"This is a string."` και στην `strB` τη συμβολοσειρά `"This is another string."` χρησιμοποιώντας την συνάρτηση `strcpy`.

```
char *strcpy(char *s1, const char *s2)
```

Η συνάρτηση `strcpy` αντιγράφει τη συμβολοσειρά `s2` στην `s1` και επιστρέφει την τιμή του δείκτη (char *) στον οποίο έκανε την αντιγραφή (σε αυτό το παράδειγμα επιστρέφει την τιμή της μεταβλητής `s1`). Χρησιμοποιώντας την `printf("%p %p %p\n", ...)` τυπώστε τους τρεις δείκτες που συμμετέχουν στην κλήση αυτής της συνάρτησης (`s1`, `s2` και την τιμή επιστροφής της `strcpy`).

1.3.3 Εκτυπώστε τις δύο συμβολοσειρές και το μήκος τους. Υπολογίστε το μήκος της `strA` μέσω της συνάρτησης `mystrlen` που υλοποιήσατε στην άσκηση 1 και το μήκος της `strB` μέσω της συνάρτησης `strlen` της C.

```
int strlen(const char *s)
```

Η συνάρτηση `strlen` επιστρέφει στο όνομα της το μήκος της συμβολοσειράς `s` (χωρίς να μετράται το τελικό `'\0'`).

1.3.4 Συγκρίνετε αλφαβητικά τις συμβολοσειρές `strA` και `strB` εκτυπώνοντας κατάλληλο μήνυμα.

```
int strcmp(const char *s1, const char *s2)
```

Η συνάρτηση `strcmp` συγκρίνει τις συμβολοσειρές `s1` και `s2` χαρακτήρα προς χαρακτήρα με βάση τους αντίστοιχους ASCII κωδικούς. Επιστρέφει:

1. `= 0`, αν οι συμβολοσειρές είναι ίδιες
2. `> 0`, αν η `s1` είναι λεξικογραφικά «μεγαλύτερη» της `s2`
3. `< 0`, αν η `s1` είναι λεξικογραφικά «μικρότερη» της `s2`

1.3.5 Προσαρτήστε τη συμβολοσειρά `strB` στο τέλος της `strA` (χρησιμοποιώντας τη συνάρτηση `mystrcat` που υλοποιήσατε παραπάνω) και εκτυπώστε το αποτέλεσμα της προσάρτησης. Στη συνέχεια, προσαρτήστε τη νέα τιμή της συμβολοσειράς `strA` στο τέλος της `strB` (χρησιμοποιώντας τη συνάρτηση `strcat` της C) και εκτυπώστε το αποτέλεσμα της προσάρτησης.

```
char *strcat(char *s1, const char *s2)
```

Η `strcat` προσαρτά / συνενώνει (concatenates) ένα αντίγραφο της συμβολοσειράς `s2` στο τέλος της `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

1.3.6 Χρησιμοποιήστε τη συνάρτηση `strtok` για να εκτυπώσετε μία προς μία τις λέξεις που εμφανίζονται στην τελική συμβολοσειρά `strB`, χωρίς τους χαρακτήρες στίξης.

```
char *strtok(char *string, const char *delim)
```

Αν το `string` δεν είναι `NULL`, η `strtok` ψάχνει στο string για την πρώτη εμφάνιση συμβολοσειράς που περιορίζεται από έναν από τους χαρακτήρες που εμφανίζονται στη συμβολοσειρά `delim`. Αν υπάρχει, αντικαθιστά τον χαρακτήρα που βρέθηκε στο string, με `'\0'` και επιστρέφει έναν δείκτη στην αρχή του `string`.

Σε κάθε επόμενη κλήση της, η `strtok` καλείται με `NULL` στο πρώτο όρισμα και συνεχίζει τη λειτουργία της από το σημείο που η τελευταία κλήση βρήκε τον χαρακτήρα διαχωρισμού. Ένα παράδειγμα χρήσης ακολουθεί με ένα quote:

```
char *p, s[] = "Little by little, one Little travels far.";
p = strtok(s, " ,");
while(p != NULL) {
  printf("%s\n", p);
  p = strtok(NULL, " ,.");
}
```

Προσθέστε το παραπάνω σε μια συνάρτηση `strtok_example` και τρέξτε την από την `main`. Στο `stdout` θα πρέπει να δείτε μια ακολουθία της μορφής:

```
Little
by
```

```
little
one
Little
travels
far
```

Άσκηση 2: Ορίσματα γραμμής εντολής - (calc.c)

2.1 Κατασκευάστε το πρόγραμμα `calc.c` που να εκτελεί απλές αριθμητικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμό, πηλίκο διαίρεσης και υπόλοιπο διαίρεσης) μεταξύ ακεραίων. Οι πράξεις που θα γίνονται να δίνονται σαν ορίσματα στη γραμμή εντολής.

Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./calc 12 + 18
30
$ ./calc 70 % 12
10
```

Ορίσματα Γραμμής Εντολής

Ορισμός της συνάρτησης `main`:

```
int main(int argc, char *argv[])
```

- Η μεταβλητή `argc` αρχικοποιείται με το πλήθος των ορισμάτων `N + 1`, τα οποία βρίσκονται στις θέσεις `argv[0]`, ..., `argv[N]` του πίνακα συμβολοσειρών `argv`. Το `argv[0]` είναι το όνομα του προγράμματος και το `argv[N+1]` είναι `NULL`.
- Η συνάρτηση `int atoi(const char *s)` επιστρέφει στο όνομά της την αριθμητική τιμή που αντιστοιχεί στην (αριθμητική) συμβολοσειρά `s`.

Άσκηση 3 (Παλιό θέμα): Πετυχαίνοντας τον στόχο - (legolas.c)

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως ορίσματα από την γραμμή εντολών έναν ακέραιο-στόχο (goal το `argv[1]`) και στην συνέχεια ένα σύνολο υποψηφίων ακεραίων (candidates) και τυπώνει όλους τους συνδυασμούς 3 υποψηφίων των οποίων το άθροισμα ισούται με τον στόχο. Η σειρά με την οποία εκτυπώνονται τα αποτελέσματα δεν έχει σημασία για την ορθότητα του προγράμματος. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o legolas legolas.c
$ ./legolas 42 19 21 3 5 12
No combination of candidates leads to 42
$ ./legolas 42 19 21 3 5 12 11
Candidates combination found: 19 + 12 + 11 = 42
$ ./legolas 42 27 25 12 31 5 26 40 34 3 18
Candidates combination found: 27 + 12 + 3 = 42
```

```
Candidates combination found: 25 + 12 + 5 = 42
Candidates combination found: 5 + 34 + 3 = 42
$ ./legolas 37372082074 9238742398 82934723 27893492387 127863435 239847289
Candidates combination found: 9238742398 + 27893492387 + 239847289 = 37372082074
```

ΠΑΡΑΡΤΗΜΑ: Αποσφαλμάτωση προγραμμάτων (Πράξη 4η)

Στο εργαστήριο 8 είδαμε ένα εργαλείο για την αποσφαλμάτωση προγραμμάτων, τον debugger gdb. Στο σημερινό εργαστήριο, θα χρησιμοποιήσουμε και πάλι debuggers για να εντοπίσουμε και να διορθώσουμε σφάλματα διαχείρισης μνήμης και, γενικότερα, λογικά σφάλματα που υπάρχουν στα προγράμματά μας.

Χρήσιμοι σύνδεσμοι:

http://sourceware.org/gdb/current/onlinedocs/gdb/

http://cgi.di.uoa.gr/~ip/debug.html

Έστω ότι θέλουμε να υπολογίσουμε το άθροισμα 1+3+5+... για τους πρώτους 20 όρους. Το παρακάτω πρόγραμμα προσπαθεί να αντιμετωπίσει αυτό το πρόβλημα.

```c
#include <stdio.h>

#define N 20

int main(int argc, char ** argv) {

  int S = 0, a = 1, i;
  for (i = 0 ; i <= N ; i++) {
    S = S+a;
    a = a+2;
  }
  printf("%d\n", S);
  return 0;
}
```

Το πρόγραμμα αυτό εκτυπώνει 441 και όχι 400 όπως θα ήταν η σωστή απάντηση. Μπορείτε να βρείτε το λάθος μέσω gdb;

Αν προσπαθήσουμε λίγο, θα διαπιστώσουμε ότι μετά το τέλος της επανάληψης, το i έχει την τιμή 21, άρα έγινε μια παραπάνω επανάληψη από όσες θέλαμε (το i παίρνει τιμές αρχίζοντας από το 0). Άρα η λύση στο πρόβλημα μας είναι να αλλάξουμε αυτή τη γραμμή κώδικα:

```c
  for (i=0 ; i<=N ; i++) {
```

σε αυτή:

```c
  for (i=0 ; i<N ; i++) {
```

Συμβουλές

Όποτε δουλεύετε με δείκτες στα προγράμματά σας, να έχετε πάντα στο νου σας τα εξής

1. Ένας δείκτης δεν αρχικοποιείται σε NULL, αλλά δείχνει σε κάποια τυχαία θέση μνήμης. Είναι πολύ σημαντικό όποτε δηλώνουμε ένα δείκτη να του δίνουμε τιμή NULL, έτσι ώστε όποτε χρησιμοποιείται, να ελέγχουμε πρώτα αν η τιμή του είναι NULL.
2. Κάθε συμβολοσειρά `char *` πρέπει να έχει αρκετό χώρο για να χωρέσει όλους τους χαρακτήρες που θέλουμε να βάλουμε συν το τελικό `\0`.
3. Όποτε χρησιμοποιούμε πίνακες, πάντα προσέχουμε να μη βγούμε έξω από τα όριά τους.

Άσκηση 4: Υπολογισμός Μισθών - (wages.c)

Το παρακάτω πρόγραμμα δημιουργεί έναν πίνακα με τυχαίους μισθούς και βρίσκει το μέσο μισθό, παραλείποντας όσα κελιά έχουν τιμή μικρότερη ή ίση του μηδενός. Όμως το πρόγραμμα δίνει segmentation fault, ενώ έχει και λογικά λάθη. Αποσφαλματώστε το με τη βοήθεια ενός debugger και διορθώστε το.

```c
#include <stdio.h>

#define N 100

int main(int argc, char ** argv) {
  int i = 0, sum = 0;
  int *wages = NULL;
  srand(N);
  for (i = 0 ; i < N ; i++)
    wages[i] = 700 + (rand()%2301) - 1000;
  while (wages[i] > 0 && i < N) {
    sum += wages[i];
    i++;
  }
  printf("Average wage is %0.2f\n", sum/N);
  return 0;
}
```