ΕΡΓΑΣΤΗΡΙΟ 9:
Συμβολοσειρές και Ορίσματα Γραμμής Εντολής

Στο εργαστήριο αυτό θα δούμε πώς ορίζονται και πώς χρησιμοποιούνται οι συμβολοσειρές 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.

Άσκηση 1: Επεξεργασία συμβολοσειρών

1.1 Υλοποιήστε τη συνάρτηση `int mystrlen(char *s)` 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`.

Άσκηση 2: Χρήση συναρτήσεων που δηλώνονται στο `string.h`

2.1 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.

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

2.1.2 Αντιγράψτε στην `strA` τη συμβολοσειρά `"This is a string."` και στην `strB` τη συμβολοσειρά `"This is another string."`.

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

Αντιγράφει τη συμβολοσειρά `s2` στην `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

2.1.3 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)
```

Επιστρέφει στο όνομα της το μήκος της συμβολοσειράς `s` (χωρίς να μετράται το τελικό `'\0'`).

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

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

Συγκρίνει τις συμβολοσειρές `s1` και `s2` χαρακτήρα προς χαρακτήρα με βάση τους αντίστοιχους ASCII κωδικούς. Επιστρέφει:

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

2.1.5 Προσαρτήστε τη συμβολοσειρά `strB` στο τέλος της `strA` (χρησιμοποιώντας τη συνάρτηση `mystrcat` που υλοποιήσατε στην άσκηση 1) 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)
```

Προσαρτά ένα αντίγραφο της συμβολοσειράς `s2` στο τέλος της `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

2.1.6 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.

Παράδειγμα χρήσης:

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

Έξοδος προγράμματος:

```
Little
by
little
one
travels
far
```

Άσκηση 3: Ορίσματα γραμμής εντολής

3.1 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:

```
% ./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[1],...,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`.

ΠΑΡΑΡΤΗΜΑ: Αποσφαλμάτωση προγραμμάτων (Πράξη 4η)

Στο εργαστήριο 8 είδαμε ένα εργαλείο για την αποσφαλμάτωση προγραμμάτων, τον debugger gdb. Στο σημερινό εργαστήριο, θα δούμε πώς μπορούμε να χρησιμοποιήσουμε τον debugger του Dev-C++ για να εντοπίσουμε και να διορθώσουμε σφάλματα διαχείρισης μνήμης και, γενικότερα, λογικά σφάλματα που υπάρχουν στα προγράμματά μας.

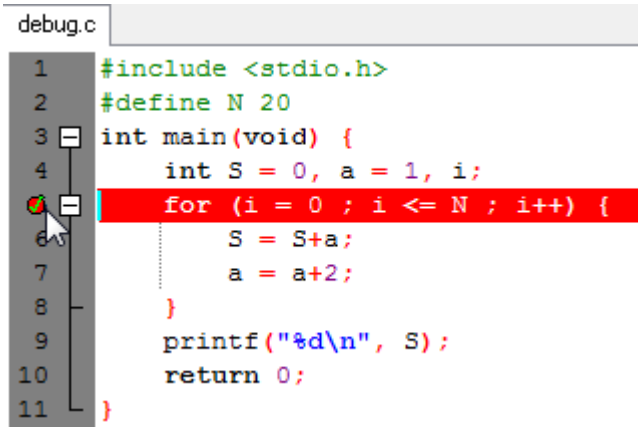
Χρήσιμοι σύνδεσμοι:
http://sourceware.org/gdb/current/onlinedocs/gdb/
http://cgi.di.uoa.gr/~ip/debug.html

Ο debugger του Dev-C++

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

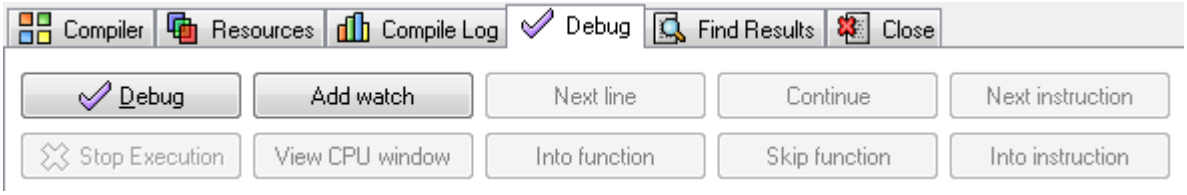
```
#include <stdio.h>
#define N 20

int main(void) {
      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. Η διαδικασία που ακολουθούμε για να εντοπίσουμε το λάθος είναι η εξής. Πρώτα, μεταγλωττίζουμε το πρόγραμμα μας. Μετά, βάζουμε ένα breakpoint σε κάποια γραμμή του κώδικα που μας ενδιαφέρει.



Αυτό το καταφέρνουμε κάνοντας κλικ δίπλα από τη γραμμή που μας ενδιαφέρει. Ένα breakpoint είναι κάποιο σημείο στο κώδικα που όταν ο έλεγχος φτάσει σε αυτό, θα σταματήσει η εκτέλεση και θα περιμένει κάποια περαιτέρω δικιά μας ενέργεια. Όπως είναι λογικό, breakpoints βάζουμε σε γραμμές του προγράμματος που είναι κώδικας και όχι δηλώσεις, σχόλια κλπ. Μπορούμε να βάλουμε όσα breakpoints θέλουμε, και τα αφαιρούμε κάνοντας πάλι κλικ δίπλα από τη γραμμή του κώδικα.

Στη συνέχεια, πατάμε το κουμπί debug και ξεκινάμε την διαδικασία της αποσφαλμάτωσης μέσα από το περιβάλλον του Dev-C++. Ο έλεγχος φτάνει στο breakpoint που θέσαμε πριν και έχουμε πια τις εξής επιλογές:



Debug: Ξεκινάει τη διαδικασία της αποσφαλμάτωσης.

Stop Execution: Αν εκτελείται ήδη αποσφαλμάτωση, τερματίζει τη διαδικασία.

Add Watch (display): Δίνοντας το όνομα κάποιας μεταβλητής, μας εμφανίζει συνέχεια στα δεξιά του παραθύρου τι τιμές παίρνει αυτή η μεταβλητή. Το ίδιο μπορούμε να καταφέρουμε αν αφήσουμε για λίγο τον κέρσορα πάνω από τη μεταβλητή αυτή.

View CPU window: Παρατηρούμε την κατάσταση του επεξεργαστή σε μια προχωρημένη προβολή (καταχωρητές και κώδικα assembly που εκτελείται), αλλά βλέπουμε και το χρήσιμο υποπαράθυρο “backtrace”, όπου περιέχεται η στοίβα εκτέλεσης και έτσι ξέρουμε ποιες συναρτήσεις δεν έχουν τερματίσει ακόμα.

Next Line (next): Συνεχίζει η εκτέλεση του προγράμματος κατά μία γραμμή, χωρίς να μπούμε στο σώμα συναρτήσεων.

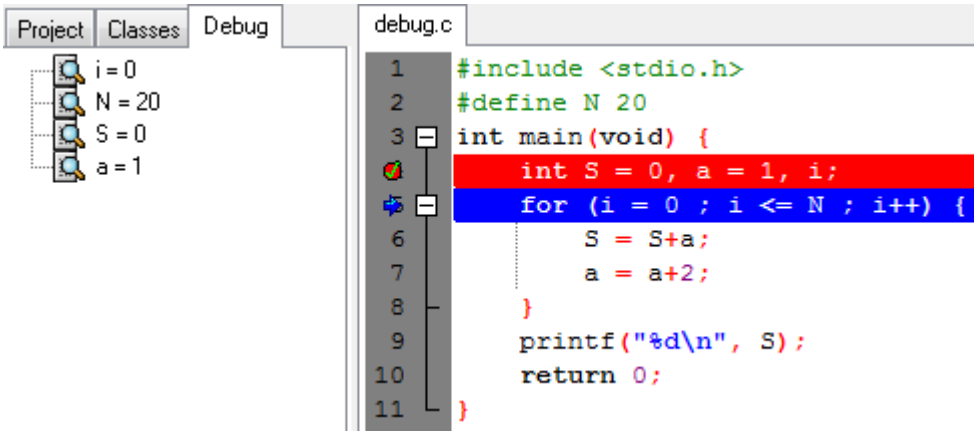
Into function (step): Συνεχίζει η εκτέλεση του προγράμματος μέχρι τη πρώτη εντολή της επόμενης γραμμής (π.χ. χρήσιμο ώστε να μη δούμε ολόκληρη την εκτέλεση μιας switch)

Continue (continue): Συνεχίζει η εκτέλεση του προγράμματος κανονικά μέχρι το επόμενο breakpoint ή, αν δεν υπάρχει κάποιο άλλο, μέχρι το τέλος του προγράμματος.

Skip function (finish): Συνεχίζει η εκτέλεση του προγράμματος κανονικά τουλάχιστον μέχρι το τερματισμό της τωρινής συνάρτησης και το επόμενο breakpoint.

Next Instruction (nexti): Εκτελείται η αμέσως επόμενη εντολή. Αν αυτή είναι κλήση συνάρτησης, γίνεται ενιαία εκτέλεση όλου του σώματος της συνάρτησης.

Into Instruction (stepi): Αν η αμέσως επόμενη εντολή είναι κλήση συνάρτησης, αντί να εκτελεστεί ενιαία, “μπαίνουμε” μέσα στο σώμα της συνάρτησης και μπορούμε στη συνέχεια να εκτελέσουμε μία-μία τις εντολές της. Αν δεν έχουμε τον πηγαίο κώδικα για την συνάρτηση (για παράδειγμα για την `printf`), δεν θα μπορέσουμε να μπούμε μέσα στο σώμα της.



Με τη βοήθεια όλων αυτών, μπορούμε να 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

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

σε αυτή

```
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

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.

```
#include <stdio.h>
#define N 100

int main(void) {
    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;
}
```