Παρατηρήσεις βαθμολόγησης lab11

Δύο αρχικές παρατηρήσεις:

1) Βλέπουμε σε πάρα πολλές ασκήσεις τη σύνταξη:

```
char str[] = {'#', '#', 'q', 'u', 'i', 't', '\0'};
```

Παρόλο που αυτό δεν είναι λάθος, δεν είναι και καλός τρόπος να αρχικοποιήσετε ένα string γιατί όσο πιο μεγάλο είναι το string τόσο αυξάνεται η πιθανότητα λάθους (όπως και έγινε σε αρκετές περιπτώσεις όπου ξεχάστηκε το '\0' με αποτέλεσμα να μη λειτουργεί η `strcmp`).

Είναι πολύ πιο πρακτικό και κατανοητό να ορίσετε: `char str[] = {"##quit"};`

2) Προσέχετε στις σύνθετες συνθήκες !!

Μια while με συνθήκη `(i<SIZE || strcmp(str, "q")!=0)` εκτελείται όσο ΕΝΑΣ από τους δύο ελέγχους είναι αληθής. Έτσι, ακόμη κι αν το i υπερβεί το SIZE, το loop θα συνεχίσει να επαναλαμβάνεται όσο το str δεν είναι "q". Το σωστό θα ήταν να είχατε &&. Δε δικαιολογείται να κάνετε ακόμη τέτοια λάθη!

ΜΕΤΑΓΛΩΤΤΙΣΗ/ΣΤΟΙΧΙΣΗ/ΚΕΝΑ:

Ισχύουν τα ίδια με προηγούμενες εργασίες.

ΕΙΣΟΔΟΣ:

Χρησιμοποιούμε sprintf για τη δημιουργία των format strings για τον delimiter και την αρχική συμβολοσειρά. Τα format strings πρέπει να είναι διαφορετικά γιατί τα strings που θα διαβαστούν έχουν διαφορετικά μεγέθη. Αν δε χρησιμοποιηθεί μεταβλητό format string, πρέπει τουλάχιστον να υπάρχει όριο στο %s ("%4s" για τον delimiter, "%63s" για το αρχικό string.)

Κατά την επανάληψη πρώτα ελέγχουμε αν το i είναι μικρότερο του ARRAY_SIZE, μετά διαβάζουμε το string και μετά ελέγχουμε αν διαβάστηκε "##quit" (το οποίο δεν πρέπει να εισάγεται στον πίνακα). Αφού περάσουμε αυτούς τους ελέγχους και μόνο τότε προχωράμε στη δημιουργία του string που ζητείται.

ΔΗΜΙΟΥΡΓΙΑ STR:

Πριν κάνουμε οτιδήποτε, αρχικοποιούμε τον πίνακα δεικτών ώστε να περιέχει NULL σε όλες τις θέσεις του (είτε με χρήση for είτε κατά τη δήλωση με ={NULL};

Προσοχή: Το `char *words[ARRAY_SIZE] = {NULL};` αρχικοποιεί σωστά κάθε ένα κελί του πίνακα ώστε να δείχνει στη διεύθυνση NULL. Το char *words[ARRAY_SIZE] = NULL; είναι συντακτικό λάθος γιατί προσπαθεί να αλλάξει τον πίνακα words σε NULL, πράγμα που δεν επιτρέπεται στη C.

Πέμπτη: Χρησιμοποιούμε `strstr(str, delim)` για να βρουμε την πρώτη εμφάνιση του delim. Αν δεν είναι NULL, και μόνο τότε, χρησιμοποιούμε `strstr(str+strlen(delim), delim)` για να βρουμε τη δεύτερη εμφάνιση.

Το μέγεθος του ενδιάμεσου string είναι `length = pos2-(pos1+strlen(delim)+1)` όπου pos1 και pos2 δείκτες στην πρώτη και δεύτερη εμφάνιση αντίστοιχα. Το +1 είναι για το '\0'. Υπάρχουν δύο τρόποι να αποσπάσουμε το ενδιάμεσο string:

- Κάνουμε το *pos2 ίσο με '\0', οπότε το ενδιάμεσο string μπορούμε να το πάρουμε με μια `strdup(pos1+strlen(delim))` η οποία θα κάνει κατάλληλο malloc και θα αντιγράψει το κομμάτι του string από τη θέση που δίνουμε ως παράμετρο μέχρι το '\0'. Αν δε χρησιμοποιήσουμε strdup, πρέπει να κάνουμε εμείς malloc και μετά strcpy.

- Δε μεταβάλλουμε το αρχικό string. Κάνουμε malloc για length χαρακτήρες και μετά strncpy για length-1 χαρακτήρες ξεκινώντας από το pos1+strlen(delim). Στο τέλος αυτών θέτουμε το '\0' (πρέπει να το κάνουμε εμείς γιατί δεν το βάζει η strncpy). Θα μπορούσαμε να αποφύγουμε την τοποθέτηση του '\0' αν αντί για malloc είχαμε χρησιμοποιήσει calloc η οποία κάνει και αρχικοποίηση της δεσμευμένης μνήμης σε μηδενικά.

Αφού δημιουργηθεί το string το αποθηκεύουμε στην επόμενη θέση του πίνακα δεικτών.

Παρασκευή: Χρησιμοποιούμε `strstr(str, delim)` για να βρουμε την εμφάνιση του delim. Αν είναι NULL, τότε κάνουμε strdup το str και το αποθηκεύουμε στην επόμενη θέση του πίνακα. Διαφορετικά, πρέπει να ξεχωρίσουμε τα δύο τμήματα. Το πρώτο είναι από την αρχή του str μέχρι το pos όπου pos η διεύθυνση που ξεκινά ο delim. Το δεύτερο είναι από το `pos+strlen(delim)` μέχρι το τέλος του αρχικού str. Υπάρχουν δύο τρόποι να τα αποσπάσουμε:

- Κάνουμε το *pos ίσο με '\0'. Με χρήση strdup (ή ισοδύναμα malloc και strcpy) δημιουργούμε αντίγραφα των str και pos+strlen(delim). Αν χρησιμοποιήσουμε malloc προσέχουμε να δεσμεύσουμε +1 θέση για το '\0'
- Δε μεταβάλλουμε το αρχικό string. Δεσμεύουμε `pos-str+1` bytes με malloc για το πρώτο τμήμα, κάνουμε strncpy `pos-str` χαρακτήρες σε αυτό ξεκινώντας από τη διεύθυνση str και προσθέτουμε το '\0' στην τελευταία θέση (πρέπει να το κάνουμε εμείς γιατί δεν το βάζει η strncpy). Για το δεύτερο τμήμα χρησιμοποιούμε είτε malloc με strcpy είτε strdup.

Αποθηκεύουμε το πρώτο τμήμα στη θέση i, και, εφόσον δεν έχει γεμίσει ήδη ο πίνακας, το δεύτερο στη θέση i+1. Προσοχή στο πώς και πότε αυξάνεται το i για το loop. Δεν είναι πάντα το ίδιο, γιατί κάποιες φορές προσθέτουμε ένα και κάποιες δύο νέα string στον πίνακα δεικτών.

FREE (ασκ. 1):

Απλή επανάληψη στον πίνακα δεικτών μέχρι είτε να τελειώσει είτε να βρούμε NULL. Σε κάθε επανάληψη κάνουμε free(words[i]); Προσοχή: δεν αρκεί να βάλουμε words[i] != NULL στη συνθήκη γιατί αν ο πίνακας είναι γεμάτος λέξεις, δε θα έχει πουθενά NULL και η επανάληψη θα συνεχίσει κι αφότου βγούμε εκτός ορίων πίνακα. Το σωστό είναι `i<ARRAY_SIZE && words[i] != NULL`. Επιπλέον, πρέπει πρώτα να ελέγχουμε αν το i είναι εντός ορίων πίνακα και μετά να προσπελάζουμε το words[i], επομένως η σειρά των δύο επιμέρους συνθηκών έχει σημασία.

ΝΕΑ ΣΥΝΑΡΤΗΣΗ (ασκ. 2):

Η συνάρτηση έχει prototype της μορφής : `int *calc_lengths(char *words[]);`

Για τον υπολογισμό μη κενών θέσεων διατρέχουμε τον πίνακα μέχρι είτε να τελειώσει είτε να συναντήσουμε NULL. Κάνουμε μια malloc που δεσμεύει τόσες θέσεις επί sizeof(int). Διατρέχουμε μια δεύτερη φορά τον πίνακα από strings, για κάθε string υπολογίζουμε το μήκος με strlen και το αποθηκεύουμε στην αντίστοιχη θέση του δυναμικού πίνακα ακεραίων. Στο τέλος επιστρέφουμε τη διεύθυνση αυτού του δυναμικού πίνακα.

PRINT (ασκ. 2):

Προσθέτουμε παράμετρο int * και αλλάζουμε την printf σύμφωνα με την εκφώνηση.

FREE(ασκ. 2):

Κάθε free αντιστοιχεί σε μια malloc. Στη δεύτερη άσκηση κάναμε μία malloc που δέσμευσε μνήμη για ολόκληρο τον δυναμικό πίνακα ακεραίων, άρα θα κάνουμε και μία free. Δεν έχει νόημα να κάνουμε κάποιου είδους επανάληψη όπως στην άσκηση 1.