תרגיל 3

הנחיות הגשה

1. העבודה היא ביחידים.
2. ההגשה היא עד ליום שלישי, בתאריך 4.12.18 בשעה 23:30.
3. הגשת התרגיל תיעשה ע"י submit. עזרה ניתן למצוא באתר:

http://help.cs.biu.ac.il/submit.htm

כדי למנוע בעיות, עדיף להגיש את הקובץ ישירות מחשבון הלינוקס שלכם.

שימו לב: העברת הקובץ דרך Windows יכולה לגרום לכך שלא יעבור אסמבלר. במקרה זה **הציון יהיה 0** ללא זכות לערעור.

4. יש לוודא שהתרגיל מתקמפל ורץ ללא שגיאות על גבי שרת ה-u2.
5. שאלות על התרגיל לשלוח בפורום הקורס (מודל – שאלות על תרגיל 3), במידה שלא עניתי לאחר מספר ימים ניתן לשלוח מייל עם קישור לשאלה ואני אענה.
6. בשורה הראשונה(!) של הקבצים אותם אתם מגישים (כל אחד מהם), לאחר TAB ורשתית (\#t) ציינו מס' ת.ז. ושם מלא. לדוג':

   #123456789 Pro Assembler

7. בהצלחה ☺

רקע כללי

ייצוג מחרוזות מסוג p-string הוא שמירת המחרוזת יחד עם האורך שלה:

```
typedef struct {
        char size;
        char string[255];
} Pstring;
```

בתרגיל זה עליכם לממש באסמבלי פונקציות ספריה שיאפשרו עבודה עם Pstring באופן דומה ל-string.h של שפת C.

מה צריך להגיש

עליכם להגיש **שלושה קבצי אסמבלי:**

1. **main.s:** קובץ שיכיל מימוש לפונקציה main (ראו "הפונקציה main").
2. **pstring.s**: source file שיכיל את המימוש לפונקציות הספריה (שיפורטו בהמשך) ללא פונקציית main.

3. **func_select.s**: קובץ שיכיל מימוש פונקציה שקוראת לפונקציית הספריה המתאימה (ראו "מבנה התוכנית").

לרשותכם מצורפים הקבצים הבאים:

1. pstring.h – מכיל את ההצהרות על הפונקציות הדרושות
2. makefile – יוצר קובץ ריצה לתוכנית
3. main-test.c – מימוש פונקציית main ב-C. קובץ זה לנוחיותכם בלבד (ראו "טיפ לתחילת העבודה"), **אינו חלק מהתרגיל**
4. makefile-test – קובץ makefile שמקמפל את התוכנית שלכם עם main-test.c במקום main.s.
התרגיל ייבדק באמצעות קימפול עם הקובץ makefile

מבנה התוכנית

פונקציית ה-main (אותה תממשו ב-main.s) מקבלת מהמשתמש שתי מחרוזות, אורכיהן ואופציה בתפריט. בונה שתי pstrings לפי המחרוזות והאורכים שהתקבלו ושולחת ל-run_func את האופציה בתפריט ואת הכתובות של ה-pstrings.
הפונקציה run_func (אותה תממשו ב-func_select.s) קוראת את האופציה בתפריט (שהתקבלה כארגומנט) ומשתמשת ב-jump table (switch-case). פירוט ערכי האופציות (cases) יופיע בהמשך.

הפונקציה main

יש לקלוט מהמשתמש מספר שלם- הוא יהיה אורך המחרוזת הראשונה, נסמן אותו ב-n_1. לאחר מכן, יש לקלוט מהמשתמש מחרוזת באורך n_1 תווים (chars)- זו המחרוזת הראשונה.
לאחר מכן יש לחזור על התהליך עבור המחרוזת השנייה- לקלוט מספר שלם חדש n_2 ולאחר מכן לקלוט מהמשתמש מחרוזת באורך n_2 תווים.
שימו לב:

1. ניתן להניח שיתקבל מספר תקין של תווים (n_1 תווים למחרוזת הראשונה ו-n_2 תווים לשנייה).
2. תו יכול להיות כל מה שמופיע בטבלת ASCII (למעט "תווי הבקרה").
3. ניתן להניח שלא יתקבל תו סיום מחרוזת מהמשתמש.
4. האחריות להוסיף תו סיום מחרוזת ('\0') היא עליכם. לכן, מדובר בסה"כ $n_1 + 1$ תווים עבור המחרוזת הראשונה ו-$n_2 + 1$ תווים עבור השניה.
5. יש לשמור את ה-Pstrings ב-stack frame של main.
6. לפני כל קריאה ל-printf או scanf, יש לאפס את rax% (חורג מהחומר שלנו, אבל נובע מכך שאנחנו שולחים להם ארגומנטים שאינם מסוג floating point). למשל:

```
movq $0,%rax
call    scanf
```

שמירת ה-Pstrings ב-stack frame של main

שימו לב שכל Pstring צריכה להישמר כך שבכתובת הנמוכה יהיה שמור אורך המחרוזת (על פני byte אחד) ובכתובות העוקבות יופיעו תווי המחרוזת. כלומר, אם יש לנו מצביע ל-Pstring, אז הערך המוצבע יהיה האורך, בכתובת העוקבת יהיה תו הראשון וכך הלאה (ראו ציור בעמ' הבא).

&Pstring + 6 →	\0
&Pstring + 5 →	o
&Pstring + 4 →	l
&Pstring + 3 →	l
&Pstring + 2 →	e
&Pstring + 1 →	h
&Pstring →	5

האופציות ב-switch-case

קבלת המספר 50

באמצעות הפונקציה pstrlen, לחשב את האורך של שתי ה-pstrings ולהדפיס את אורכיהן. פורמט ההדפסה:

"first pstring length: %d, second pstring length: %d\n"

קבלת המספר 51

יש לקלוט מהמשתמש שני תווים (chars)- התו הראשון יהיה התו שצריך להחליף (oldChar) והתו השני יהיה התו החדש (newChar). באמצעות הפונקציה replaceChar, להחליף בשתי ה-pstrings כל מופע של oldChar ב-newChar.

לאחר ההחלפה, יש להדפיס את שתי ה-pstrings בפורמט ההדפסה:

"old char: %c, new char: %c, first string: %s, second string: %s\n"

קבלת המספר 52

יש לקלוט מהמשתמש שני מספרים (שלמים)- המספר הראשון יהיה אינדקס התחלה והשני אינדקס סיום. לאחר מכן, לקרוא לפונקציה pstrijcpy, כאשר j הוא אינדקס הסיום, i אינדקס ההתחלה, src הוא המצביע ל-pstring השנייה ו-dst המצביע ל-pstring הראשונה.

לאחר ההעתקה, יש להדפיס את ה-pstring הראשונה (destination) בפורמט ההדפסה:

"length: %d, string: %s\n"

ולאחר מכן להדפיס את ה-pstring השנייה (source) לפי הפורמט הנ"ל.
לשים לב- את ההודעות הנ"ל מדפיסים גם אם pstrijcpy הדפיסה הודעת שגיאה (ראו הסבר על pstrijcpy).

קבלת המספר 53

באמצעות הפונקציה swapCase, להחליף בכל pstring כל אות אנגלית גדולה (A-Z) באות אנגלית קטנה (a-z), ולהחליף כל אות אנגלית קטנה באות אנגלית גדולה.
לאחר ההחלפה, יש להדפיס את ה-pstring הראשונה ואחריה את ה-pstring השנייה בפורמט ההדפסה:

"length: %d, string: %s\n"

קבלת המספר 54

יש לקלוט מהמשתמש שני מספרים (שלמים)- המספר הראשון יהיה אינדקס התחלה והשני אינדקס סיום. לאחר מכן, לקרוא לפונקציה pstrijcmp, כאשר j הוא אינדקס הסיום, i אינדקס ההתחלה, pstr1 הוא המצביע ל-pstring הראשונה ו-pstr2 המצביע ל-pstring השנייה.
לאחר ההשוואה, להדפיס את תוצאת ההשוואה בפורמט ההדפסה:

"compare result: %d\n"

לשים לב- את ההודעה הנ"ל מדפיסים גם אם pstrijcmp הדפיסה הודעת שגיאה (ראו הסבר על pstrijcmp).

בכל מקרה לאחר ביצוע הפעולה יש לסיים את התוכנית בצורה מסודרת.

במידה ו-run_func קיבלה מספר אחר, יש להדפיס למשתמש את ההדפסה:

"invalid option!\n"

הפונקציות שצריך לממש ב-pstring.s

char pstrlen(Pstring* pstr)

הפונקציה מקבלת מצביע ל-Pstring ומחזירה את אורך המחרוזת.

Pstring* replaceChar(Pstring* pstr, char oldChar, char newChar)

הפונקציה מקבלת מצביע ל-Pstring ושני chars, ומחליפה כל מופע של oldChar ב-newChar. הפונקציה מחזירה את המצביע pstr (לאחר שינוי המחרוזת).
אפשר להניח ש-oldChar ו-newChar לא יהיו '\0'.

Pstring* pstrijcpy(Pstring* dst, Pstring* src, char i, char j)

הפונקציה מקבלת שני מצביעים ל-Pstring, מעתיקה את תת-המחרוזת src[i:j] (כולל) לתוך dst[i:j] (כולל) ומחזירה את המצביע ל-dst.
אפשר להניח שהאורך של **dst** לא ישתנה לאחר ההעתקה.
אם האינדקסים i או j חורגים מגבולות src או dst, אין לשנות את dst ויש להדפיס את ההודעה:

"invalid input!\n"

Pstring* swapCase(Pstring* pstr)

הפונקציה מקבלת מצביע ל-Pstring, הופכת כל אות אנגלית גדולה (A-Z) לאות אנגלית קטנה (a-z) ולהיפך – הופכת כל אות אנגלית קטנה (a-z) לאות אנגלית גדולה (A-Z).
שימו לב שבמחרוזת יכולים להופיע גם תווי ASCII שאינם אותיות. אין לשנות תווים אלו.

int pstrijcmp(Pstring* pstr1, Pstring* pstr2, char i, char j)

הפונקציה מקבלת שני מצביעים ל-Pstring ומשווה בין בין pstr1->str[i:j] (כולל) לבין pstr2->str[i:j] (כולל):

1. הפונקציה מחזירה 1 אם pstr1->str[i:j] גדולה לקסיקוגרפית (לפי ASCII) מ-pstr2->str[i:j]
2. הפונקציה מחזירה 1- אם pstr2->str[i:j] גדולה לקסיקוגרפית מ-pstr1->str[i:j]
3. הפונקציה מחזירה 0 אם pstr1->str[i:j] ו-pstr2->str[i:j] זהות

אם האינדקסים i או j חורגים מגבולות src או dst, ערך ההחזרה הוא (2-) ויש להדפיס את ההודעה:

"invalid input!\n"

לשים לב שיתכן ו-pstr1,pstr2 נבדלות באורך, אבל תת-המחרוזת בין התו ה-i לתו ה-j (כולל) בכל אחת מהן זהה.

טיפ לתחילת העבודה

מומלץ לממש את התרגיל בהדרגה. כלומר:

1. תיעזרו בקובץ main-test.c בשלב הראשון. כך תדעו שהמחרוזת נקלטה באופן נכון ותוכלו לבדוק בהצלחה את הפעולות שתבצעו עליהן (לפי func_select.s ו-pstring.s). **שימו לב** שעליכם תחילה לשנות את שם הקובץ makefile-test ל-makefile, זאת כדי להפעיל דווקא אותו באמצעות הפקודה make. הערה: בקובץ main-test.c יש מימוש שבו יש גבול למחסנית (כיוון שזו מגבלה של c), בתוכנית שלכם זה לא צריך להיות המצב ואתם אמורים לממש את זה באופן דינאמי.
2. תתחילו לכתוב את הקוד של func_select.s מבלי לקרוא בינתיים לפונקציות מתוך pstring.s – באמצעות הדיבאגר (ראו הוראות באתר) תוכלו לוודא שהגעתם למקום הנכון בקוד לפי ה- jump table.
3. תקלטו ערכים באמצעות scanf ותדפיסו אותם באמצעות printf.
4. תממשו את הפונקציות מתוך pstring.s ותקראו להן מ-func_select.s (תבדקו כל פונקציה לאחר המימוש שלה).
5. לאחר שווידאתם שהקוד ב-func_select.s, pstring.s תקין, תממשו את פונקציית main באסמבלי בקובץ main.s. תחליפו את הקובץ makefile-test בקובץ makefile (מתוך הקבצים המצורפים).

הערות

1. **אין להשתמש בפונקציות מתוך string.h**.
2. הפונקציות החיצוניות היחידות המותרות בשימוש הן פונקציות המערכת printf ו-scanf.
3. אין לבצע הקצאות זיכרון דינמיות – עבור זיכרון נוסף, ניתן להשתמש ב-stack.
4. בזמן הבדיקה יוכנסו מספר קלטים שונים ויבדקו גם מקרי קצה, אך ניתן להניח כי כל הקלטים שיוכנסו יהיו בהתאם למה שהוגדר בתרגיל.
5. אפשר להניח שבכל ה-pstrings שיתקבלו, אורך המחרוזת יהיה שווה לשדה האורך שלה (למשל, אם הוכנסה המחרוזת "hello", אז בהכרח הערך של שדה האורך שלה יהיה 5).
6. יש להקפיד לממש את דרישות התרגיל, בדיוק כפי שהוגדרו (לא להשתמש ברצף של תנאי if במקום ב-switch, לדוגמא).
7. תוכנית שלא תעבור אסמבלר ו/או ,תיכשל בבדיקה תקבל ציון סופי 0 – לא תינתן זכות לערעורים על כך.
8. יש צורך בהערות **משמעותיות** כל 1-5 שורות וכמובן בכל פקודה לא טריוויאלית. מתכנת חיצוני (למשל, הבודק של הקורס) שמסתכל על הקוד שלכם צריך להבין בקלות את מהלך התוכנית.
9. הקפידו על סידור הטקסט: TAB לפני הפקודה עצמה (אך לא לפני label), tab בין הפקודה לארגומנטים שלה ו-tab לפני הערות. בצורה זו שמות כל הפקודות (addq, movq, subq,...) נמצאות אחת מתחת לשנייה, רצף הארגומנטים של כל פקודה נמצא אחד מתחת לשני וכנ"ל לגבי ההערות. ראו ב-Hello.s לדוגמא.
10. חובה להשתמש בכל אוגר לפי הכללים החלים עליו שנלמדו בקורס (caller save, callee save וכו').
11. יש להקפיד על מוסכמות העברת הארגומנטים לפונקציות בהתאם ל-x86-64 (העברת 6 הארגומנטים הראשונים דרך הרגיסטרים המתאימים וכו').
12. בכתיבת כל פונקציה יש להקפיד על הפעולות הידועות בתחילתה: הקצאת frame בגודל המתאים (במידה ויש צורך בכך), שמירת האוגרים השמורים בהם יעשה שימוש, וכל הפעולות ההפוכות לפעולות הנ"ל בסוף הפונקציה.
13. הקוד צריך להיות יעיל וקריא.

דוגמת הרצה

5
hello
5
world
50
first pstring length: 5, second pstring length: 5

5
hello
3
bye
51
e z
old char: e, new char: z, first string: hzllo, second string: byz

5
hello
3
bye
51
z a
old char: z, new char: a, first string: hello, second string: bye

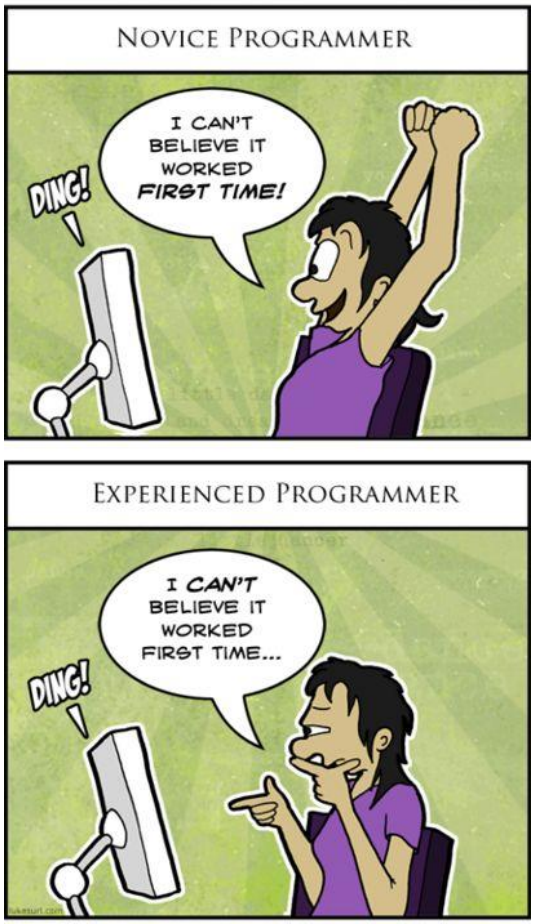
---

5
hello
5
world
52
1
4
length: 5, string: horld
length: 5, string: world

5
He@lo
5
WORLD
53
length: 5, string: hE@LO
length: 5, string: world

5
hello
5
world
54
1
10
invalid input!
compare result: -2

5
hello
5
world
78
invalid option!



בהצלחה!