תרגיל בית flex & bison (או flex & recursive descent parser) סמסטר ב 2024

הגשה דרך moodle

יש להגיש בזוגות.

מה צריך להגיש: קובץ zip הכולל את הקוד שאתם כתבתם ודוגמא לקלט עם הפלט המתאים. אין להגיש את הקבצים שיצרו flex & bison.

יש להגיש 2 גרסאות של התכנית: גרסה אחת תשתמש ב- flex וב-parser שנוצר ע"י bison. גרסה שניה תשתמש ב- recursive descent parser (וב-flex). את הגרסה הראשונה הגישו בתיקיה שתקרא bottom_up. את הגרסה השנייה הגישו בתיקיה שתקרא top_down.

יש לכתוב תכנית שהקלט והפלט שלה הם:

קלט: כמו בתרגיל הבית הראשון (על flex):
הקלט כולל רשימה של מקצועות ספורט. עבור כל מקצוע מופיע בקלט שם המקצוע ורשימה של השנים בהם התחרו באותו מקצוע באולימפיאדה.

הקלט יופיע בקובץ. שם הקובץ יהיה ה- command line argument היחיד של התוכנית.

פלט (שיכתב ל- standard output): שמות המקצועות שהופיעו בשבע אולימפיאדות לפחות.
בנוסף לכך בפלט יופיע המספר הממוצע של אולימפיאדות בהן מופיע מקצוע.
(יש דוגמא מפורטת בהמשך).

דרך החישוב של הממוצע המבוקש: יש לחשב עבור כל מקצוע בכמה אולימפיאדות הוא הופיע. ואז הממוצע הוא הסכום של המספרים שחושבו מחולק במספר המקצועות.

בחישוב של מספר האולימפיאדות בהן מופיע מקצוע יש להניח:
האולימפיאדה הראשונה (בזמנים מודרניים) היתה בשנת 1896. (זה רלוונטי עבור מקצועות שהופיעו בכל האולימפיאדות).
האולימפיאדה האחרונה בינתיים היתה בשנת 2021. אבל, למען הפשטות נניח בהמשך שהאולימפיאדה האחרונה התקיימה בשנת 2020.

למען הפשטות ניתן להתעלם מהעובדה שחלק מהאולימפיאדות בוטלו (עקב שתי מלחמות עולם).

למשל אחת משורות הקלט בדוגמא שמופיעה למטה היא

```
[sport] "Tug Of War" [years] 1900 to 1920
```

אז אפשר לחשב את מספר האולימפיאדות בהן התחרו ב- Tug Of War
("משיכת חבל") כך: ((1920-1900)/4) + 1 = 6
(האמת היא שמספר האולימפיאדות הוא 5 בלבד כי האולימפיאדה בשנת 1916 בוטלה).

2020
!
1896

<u>דוגמא:</u>

הנה דוגמא לקלט:

input:

```
Olympic Sports
<sport> "Archery" <years> 1900-1908, 1920, since 1972

<sport> "Athletics" <years> all

<sport> "BasketBall" <years> since 1936

<sport> "Tug Of War" <years> 1900 to 1920

<sport> "Karate" <years> 2020
```

בדוגמא זו הפלט צריך להיות:

```
sports which appeared in at least 7 olympic games:
Archery
Athletics
Basketball

average number of games per sport:  15.60
```

הנה החישוב עבור הדוגמא:

ספורט	מספר אולימפיאדות
Archery	(1908-1900)/4 + 1 + 1 + (2020–1972)/4+1 = 17
Athletics	(2020-1896)/4 + 1 = 32
Basketball	(2020-1936)/4 + 1 = 22
Tug Of War	(1920–1900)/4 + 1 = 6
Karate	1
	סכום: 17+32+22+6+1 = 78 ממוצע: 78/5 = 15.60

הנה תיאור סוגי האסימונים (כפי שהופיע בתרגיל הבית הראשון):

<u>סוגי אסימונים</u>

סוג אסימון (token type)	מה מופיע בקלט
TITLE	Olympic Sports
SPORT	<sport>
YEARS	<years>
NAME	שם הספורט מוקף בגרשיים למשל "Archery"
YEAR_NUM	השנה. למשל 2016. השנה המוקדמת ביותר שעשויה להופיע בקלט היא 1896 המאוחרת ביותר היא 2020.
COMMA	פסיק
HYPHEN	מקף (hyphen) או המחרוזת to
SINCE	since
ALL	all

<u>דקדוק לתאור הקלט (בפורמט של bison)</u>

בהתאם למוסכמה של bison -- אסימונים כתובים כאן באותיות גדולות, ומשתנים כתובים באותיות קטנות. (גם תווים המוקפים בגרש בכל צד כמו למשל ',' הם אסימונים).

שימו לב שבסוף כל כלל גזירה (או מספר כללי גזירה המופרדים ע"י |) מופיע נקודה פסיק בהתאם לפורמט של bison.

(האסימון שנקרא למעלה COMMA מופיע כאן כ- ',' . השתמשו בסימון שנוח לכם).

סימון של bison: כשרשום למשל list_of_sports: %empty; הכוונה היא שכלל הגזירה אומר ש- list_of_sports גוזר את המילה הריקה.

```
input: TITLE list_of_sports;

list_of_sports: list_of_sports sport_info;

list_of_sports: %empty;

sport_info: SPORT NAME YEARS list_of_years;

list_of_years: list_of_years ',' year_spec;
list_of_years: year_spec;

year_spec: YEAR_NUM |
           ALL      |
           YEAR_NUM HYPHEN YEAR_NUM |
           SINCE YEAR_NUM
          ;
```

<u>לגבי recursive_descent_parser</u>:

הדקדוק הנתון אינו LL(1) בגלל שיש בו רקורסיה שמאלית (בכללים של list_of_sports ושל list_of_years) וגם בגלל שני הכללים של year_spec

בהם האסימון הנגזר הראשון הוא YEAR_NUM. אבל למרות זאת לא קשה לכתוב recursive descent parser:
את הכללים עם רקורסיה שמאלית ניתן להחליף ברקורסיה ימנית. או לחילופין ראו
בדוגמאות ל- recursive descent parser שיש במודל (ומוזכרות בהמשך) כיצד
ניתן לטפל בסדרות בעזרת לולאת while. את הבעיה עם year_spec ניתן לפתור ע"י כך שבמקרה
שהפונקציה שעושה ניתוח תחבירי ל- year_spec רואה את האסימון YEAR_NUM
בקלט היא תקרא את האסימון שבא אחריו כדי להחליט באיזה משני הכללים הבאים להשתמש:
year_spec -> YEAR_NUM | YEAR_NUM TO YEAR_NUM

הערות

עליכם להחליט באיזה ערכים סמנטיים להשתמש. אין להשתמש במשתנים גלובליים (כדי לתרגל את השימוש בערכים סמנטיים).

תזכורת: הכנת תוכנית בעזרת flex & bison.
(ההערות מתיחסות ל- Windows אבל דבר דומה יעבוד על Linux)

Bottom up

נניח שברשותנו קובצי קלט ל- flex ול- bison שהכנו בעזרת text editor (למשל Notepad++). נקרא לקבצים olympics.lex ו- olympics.y.

נריץ את הפקודות הבאות בחלון המריץ את cmd.exe (או בחלון המריץ shell של MinGW או משהו דומה לכך).

1. מריצים את flex

```
flex olympics.lex
```

נוצר קובץ lex.yy.c (שבו הפונקציה yylex).

2. מריצים את bison עם האופציה d–

```
bison -d olympics.y
```

bison יצור שני קבצים: olympics.tab.c ו- olympics.tab.h. (את השני הוא יצור בגלל האופציה d–).

הערה: בקובץ olympics.tab.c תמצא הפונקציה yyparse.

בקובץ olympics.tab.h ימצאו הגדרות של האסימונים, של ה- union והכרזה של yylval. אנו יוצרים את הקובץ הזה כדי שניתן יהיה לעשות לו include במקום המתאים ב- olympics.lex כדי שהפונקציה yylex תכיר את ההגדרות של האסימונים ושל yylval שכן היא משתמשת בהם.

הערה נוספת: אין חשיבות לסדר שבו מבצעים את שני הצעדים הראשונים כלומר ניתן להריץ קודם את bison ולאחר מכן את flex.)

3. יש לקמפל את קובצי ה- C ש- flex & bison יצרו עבורנו.

(כמובן שאם התוכנית שלנו כוללת קבצים נוספים יש לקמפל גם אותם).
לצורך כך ניתן להשתמש בכל קומפיילר לשפת C.

אם נשתמש בקומפיילר gcc (קומפיילר פופולרי של GNU) הפקודה היא:

```
gcc -o olympics.exe  lex.yy.c olympics.tab.c
```

כאן האופציה o- מציינת את שם הקובץ שהוא התוצר של הקומפילציה (במקרה זה שם הקובץ הוא olympics.exe).

4. נכין קובץ טקסט שנקרא לו test_olympics.txt ובו נכתוב קלט לדוגמא למשל

```
Olympic Sports
<sport> "Archery" <years> 1900-1908, 1920, since 1972

<sport> "Athletics" <years> all
          . . .
```

נריץ את הפקודה

```
olympics.exe  test_olympics.txt
```

והפלט יהיה:

Out Put:

```
sports which appeared in at least 7 olympic games:
Archery
Athletics
Basketball

average number of games per sport:  15.60
```

<u>הכנת תוכנית עם recursive descent parser (ועם flex כדי לייצר את המנתח הלקסיקלי):</u>

Top Down

נניח שקובץ הקלט ל- flex נקרא olympics.lex וששאר הקוד שלנו (כולל הקוד של ה- recursive descent parser) נמצא בקבצים olympics.c ו- olympics.h. בקובץ האחרון נשים הגדרות והכרזות משותפות ל- lexer (yylex) ול- parser: הגדרות של סוגי האסימונים, הכרזה של משתנה גלובלי שבו yylex יכתוב את הערך הסמנטי של האסימון שהוא מחזיר (משתנה בעל תפקיד דומה ל- yylval של bison) ואולי דברים נוספים.

1. מריצים את flex

```
flex olympics.lex
```

נוצר קובץ lex.yy.c (שבו הפונקציה yylex).

2. יש לקמפל את קובצי ה- C שלנו (כולל הקובץ ש- flex כתב עבורנו).
לצורך כך ניתן להשתמש בכל קומפיילר לשפת C.

אם נשתמש בקומפיילר gcc הפקודה היא:

```
gcc -o olympics.exe  lex.yy.c olympics.c
```

כאן האופציה o- מציינת את שם הקובץ שהוא התוצר של הקומפילציה (במקרה זה שם הקובץ הוא `olympics.exe`).

ההמשך (הרצת התוכנית על קובץ קלט) כמו בדוגמא (שלב 4) שמופיעה למעלה בתיאור של הכנת תוכנית בעזרת flex & bison.

<u>בכל מקרה מומלץ להשתמש ב- makefile</u>

<u>התוכנות של flex & bison נמצאות ב- moodle בתיקיה על bison</u>
(אלו מתאימות להרצה על Windows).

דוגמאות לתוכניות
בתיקיה של תרגילי הבית סמסטר 2024 ב יש שתי תכניות לדוגמא

לכל אחת מהתכניות האלו יש 2 גרסאות : אחת משתמשת ב- bison (וב- flex) והשנייה משתמשת ב- recursive descent parser (וב- flex).

בנוסף לכך אפשר להסתכל בפתרונות של בחינות (בתיקית הבחינות) מהשנים האחרונות. השאלה הראשונה בכל בחינה עוסקת ב- flex & bison.

בהצלחה!