

הגשה דרך moodle

אפשר להגיש בזוגות או בשלוש (אבל שלשות צריכים לעשות יותר כפי שמפורט בהמשך).

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

את התוכנית ניתן לכתוב תוך שימוש ב-flex וב-bison או לחילופין, ניתן לכתוב recursive descent parser במקום להשתמש ב-parser הנוצר בעזרת bison. גם במקרה השני ניתן להשתמש ב-flex כדי לייצר את המנתח הלקסיקלי. שלשות צריכות להכין את שתי הגרסאות: גם את הגרסה עם bison וגם את הגרסה עם recursive descent parser.

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. (לצורך החישובים ניתן להניח שהאולימפיאדה האחרונה התקיימה ב-2020 במקום ב-2021).

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 through 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).

דוגמא:

הנה דוגמא לקלט :

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 through 1921

<sport> [Karate] <years> 2021

בדוגמא זו הפלט צריך להיות:

sports which appeared in at least 7 olympic games:

Archery

Athletics

Basketball

average number of games per sport: 15.20

הנה החישוב עבור הדוגמא:

(בחישובים נוח להתייחס לאולימפיאדה של 2021 כאילו התקיימה ב- 2020)

מספר אולימפיאדות	ספורט
$(1908-1900)/4 + 1 + 1 + (2016-1972)/4 + 1 = 16$	Archery
$(2020-1896)/4 + 1 = 32$	Athletics
$(2020-1936)/4 + 1 = 22$	Basketball
$(1920-1900)/4 + 1 = 6$	Tug Of War
1	Karate
סכום:	
$16+32+22+6+0 = 76$	
ממוצע:	
$76/5 = 15.20$	

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

סוג אסימון (token type)	מה מופיע בקלט
TITLE	Olympic Sports
SPORT	<sport>
YEARS	<years>
NAME	שם הספורט מוקף בסוגריים מרובעות למשל [Archery]
YEAR_NUM	השנה. למשל 2016. השנה המוקדמת ביותר שעשויה להופיע בקלט היא 1896 המאוחרת ביותר היא 2021. השנה 2020 לא אמורה להופיע בקלט.
COMMA	פסיק
THROUGH	מקף (hyphen) או המחרוזת through
SINCE	since
ALL	all

### דקדוק לתאור הקלט (בפורמט של bison)

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).

```
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 THROUGH YEAR_NUM |
          SINCE YEAR_NUM
          ;
```

### לגבי recursive descent parser

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

בדוגמא ל- recursive descent parser שיש במודל (ומוזכרת בהמשך) כיצד ניתן לטפל בסדרות בעזרת לולאה. 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 THROUGH YEAR\_NUM

## הערות

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

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

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 ושל 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:

```
sports which appeared in at least 7 olympic games:
```

```
Archery
```

```
Athletics
```

```
Basketball
```

```
average number of games per sport: 15.20
```

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

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.

בכל מקרה מומלץ להשתמש ב- makefile למי שמכיר. (זו דרך להקל על בנית התוכנית שלכם).

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

**דוגמאות לתוכניות שהוכנו בעזרת flex & bison:**

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

יש באתר הקורס דוגמא גם לתכנית עם **recursive descent parser** ראו בתיקיה top down parsing את "דוגמא ל- recursive descent parser שמחשב ערך לפי הקלט (שירים)"

**בהצלחה!**