



פרויקט – חלק 2

נמשיך במלאת בניית הקומpileר והפעם בשימוש באסימונים של שפת -- C על מנת לבנות עץ גזירה.
בתרגילים הבאים נשתמש מנתח תחבירי הכלול פועלות סמנטיות בסיסיות לצורך בניית העץ. המנתח התחבירי
יקבל קלט של אסימונים מהנתח הלקסיקלי שמייחשתם בחלק 1 וידפיס את העץ כפלט.

דקדוק השפה -- C

יש לשים לב להבדל בין DEC (Declaration) ל-(DEF (Definition) בכללי הגירה!

```

PROGRAM      → FDEFS
FDEFS       → FDEFS FUNC_DEF_API BLK | FDEFS FUNC_DEC_API | ε

FUNC_DEC_API → TYPE id ( ) ; |
                  TYPE id ( FUNC_ARGLIST ) ;

FUNC_DEF_API → TYPE id ( ) |
                  TYPE id ( FUNC_ARGLIST )

FUNC_ARGLIST → FUNC_ARGLIST , DCL | DCL

BLK          → { STLIST }
DCL          → id : TYPE | id , DCL
TYPE         → int | float | void
STLIST        → STLIST STMT | ε
STMT          → DCL ; | ASSN | EXP ; | CNTRL | READ | WRITE | RETURN |
                  BLK
RETURN        → return EXP ; | return;
WRITE         → write ( EXP ) ; | write ( str ) ;
READ          → read ( LVAL ) ;
ASSN          → LVAL assign EXP ;
LVAL          → id
CNTRL         → if BEXP then STMT else STMT | if BEXP then STMT |
                  while BEXP do STMT

BEXP          → BEXP or BEXP | 
                  BEXP and BEXP | 
                  not BEXP | 
                  EXP relop EXP | 
                  ( BEXP )

EXP           → EXP addop EXP | 
                  EXP mulop EXP | 
                  ( EXP ) | 
                  ( TYPE ) EXP |

```

המשך הדקדוק בעמוד הבא



```

id | 
NUM | 
CALL

NUM          → integernum | realnum

CALL          → id ( CALL_ARGS )
CALL_ARGS:   → ε |
              POS_ARGLIST |
              NAMED_ARGLIST |
              POS_ARGLIST , NAMED_ARGLIST

POS_ARGLIST  → EXP |
                  POS_ARGLIST , EXP
NAMED_ARGLIST → NAMED_ARG |
                  NAMED_ARGLIST , NAMED_ARG
NAMED_ARG    → id : EXP

```

בכל תחביר, משתני השפה הינם המשתנים המופיעים בחלק השמאלי של החוקים (סומנים באותיות גדולות). הטרמינלים הינם האסימונים שהוגדרו בחלק 1 של הפROYיקט (סומנים באותיות קטנות ותווים בודדים). המשתנה התחيلي הינו המשתנה של חוק הגירה הראשון בראשימה, בולם, במקרה שלנו: PROGRAM .

המשימה:

1. כתבו מנתה תחבירי אשר מקבל תוכנית בשפת--C, גוזר את התוכנית ומדפיס את עץ הגזירה שלה. מנת לעשות זאת עליכם לบทוב קובץ `z` עבור הכליל `Bison` ולעדכן את קובץ ה- `Flex` אשר כתבתם בתרגיל הקודם, על מנת להעיבר את האסימונים לנתה תחבירי, כפי שלמדו.
2. שלב הניתוח תחבירי יבצע את בניית עץ הגזירה באמצעות כלים סמנטיים שיוצמדו לחוקי הגירה, תוך שימוש במבנה נתונים של עץ אשר כל צומת בו מוגדר באופן הבא:

```

typedef struct node {
    char *type; // Syntax variable or token type for tokens
    char *value; // Token value. NULL for syntax variables
    struct node *sibling;
    struct node *child;
} Node, *NodePtr;

```

השדה `type` מכיל את סוג האסימון, כמו גדר בחלק 1 של הפROYיקט. השדה `value` מכיל את ערך העזר של האסימון (בדרא"ב הלקסמה), עבור האסימונים הרלוונטיים, בהתאם להגדרות בחלק 1 של הפROYיקט. עבור אסימונים ללא ערך עזר שדה `value` יהיה NULL.

ראו ציר בעמוד הבא:

הפקולטה להנדסת חשמל

באמצעות צומת זה נוכל ליצג עץ עם שורש שהוא שלושה צאצאים כך:

```

parent, root
+-----+
| type   |
+-----+
| value  |
+-----+
| sibling |
+-----+
| child   | ===+
+-----+   |           +-----+           +-----+
                  |           |           |
                  V           V           V
+-----+   |           +-----+           +-----+
| type   |   |           | type   |   |           | type   |
+-----+   |           +-----+   |           +-----+
| value  |   |           | value  |   |           | value  |
+-----+   |           +-----+   |           +-----+
| sibling | ===+   | sibling | ===+   | sibling | ===+
+-----+   +-----+   +-----+   +-----+
| child   |   | child   |   | child   |   V
+-----+   +-----+   +-----+   +-----+   NULL

```

שימוש לבני הצאצאים של צומת מסויים מוגדרים בראשימת צמות הצעדים החל מהצעדה השמאלית ביותר.

3. בסיום בניית העץ על המנתה להדפיס את עץ הגזירה. על מנת להבטיח אחידות בפלט העץ, מסופקת לכם הפונקציה `dumpParseTree` המדפיסה את העץ. הפונקציה מקבלת כי קיים מצביע גלובלי בשם `parseTree` מסוג `*ParserNode`, המצביע בעטום הגזירה לzonot השרוש של עץ הגזירה הנוכחי. בהתאם, על המנתה לבנות את העץ כך ששורשו מצביע על ידי משתנה גלובלי בשם `parseTree`.

4. עליכם לספק, כמו בחלק 1 של הפרויקט, קובץ makefile לבניית קובץ ההרצה של המנתה התchapיר. שום קובץ הריצה שייצר חייב להיות **part2**. קובץ ריצה זה יופעל כמו בחלק 1, בלומר, יקבל את הקלט מהקלט הסטנדרטי, או מוקבע באמצעות redirection.

5. בבנית מנהה תחבירי עם `bison` לא מלנקגים עם הספריה `fi`, כפי שעשינו בחולק 1 של הפROYיקט, וכן יש צורך למשם פונקציית `main`. עם חומרី העזר לחולק 2, מסופקת לכם פונקציית `main` למנהח התחבירי.
פונקציה זו מפעילה את פונקציית המנתחה (`yyparse()` ולאחר מכן קוראת ל-`dumpParseTree` להדפסת עץ הבניתוח, במידה והצליח.

הتمודדות עם שגיאות והודעות שגיאה:

אין הנחת קלט תקין בפרויקט, כלומר יש להתמודד עם שגיאות. במקרה של גלוּ שגיאה בזמן הניתוח, יש לעצור את המנתח, להוציא לפולט הסטנדרטי הודיעת שגיאה, ולצאת מהמנתח עם החזרת קוד יציאה/שגיאה כמפורט להלן:

1. עבר שגיאה לקסיקלית, קוד שגיאה 1 והודעה במבנה הבא:

Lexical error: '<lexeme>' in line number <line_number>

2. עבד שגיאה תחבירית, קוד שגיאה 2 והודעה במבנה הבא:



Syntax error: '<lexeme>' in line number <line_number>

באשר <lexeme> הינה הלקסמה הנוכחית בעת השגיאה.

במקרה של שגיאה אין להוציא כל פלט אחר מלבד הודעה השגיאה. למשל, רק במקרה של סיום מוצלח של הניתוח יודפס עץ הגירה.

אין צורך לטפל בשגיאות סמנטיות אלא רק בשגיאות הנובעות מהනיתוח התחבירי.

* עימם בתייעוד של Bison לגבי טיפול בשגיאות תחביריות, ובפרט בפרק בשם:

The Error Reporting Function yyerror.

הנחיות נוספות:

- הדקוק הוא רב משמעי ביחס לחלק הממשתנים. אין לשנות את הדקוק כדי לפתור את הקונפליקטים. יש להגדיר עדיפויות אסוציאטיביות (של Bison) עבור האסימונים כדי לפתור את הקונפליקטים. יש לישם קדימות ואסוציאטיביות במקובל ב-C/C++. ניתן להיעזר בסיכון הקדימות בקישור הבא:
http://en.cppreference.com/w/cpp/language/operator_precedence
- פונקציות העדר לבניית המנתה של עץ הניתוח, הדפסתו וה-main כנ"ל מסופקות לכם בקבצים part2_helpers.c/h . יש לכלול קבצים אלו בהגשה כך שניתן לבנות את המנתה ללא צורך בהוספה קבצים נוספים באמצעות פקודה make.
- לחומרו התרגיל מצורפות דוגמאות לתוכניות קלט והפלט המצופה בעובן.
- בנוסף לדוגמאות המסופקות, מומלץ ליצור עוד קליטים לבדיקת המנתה שלכם – גם קלט שבשפה (good cases) וגם קלט שגוי (bad cases).
- פרטיהם נוספים על שימוש ברכי-h Bison מצויים במצגת התרגול ובקישורים באתר הקורס.



הוראות ההגשה

- מועד אחרון להגשתה: יום א' 28/12/2025 בשעה 23:55.
- שימוש-לב למדייניות בנוגע לאישורים בהגשתה המפורסמת באתר הקורס. במקרה של נסיבות המצדיקות אישור, יש לפנות מראש לצוות הקורס לティום דחיית מועד ההגשתה.
- ההגשתה בזוגות. הגשה בבודדים תתקבל רק באישור מראש מצוות הקורס.
- יש להגיש بصورة מקוונת באמצעות אתר ה-e-Moodle של הקורס, מחשבינו של אחד הסטודנטים. הקיפויו לוודא כי העלייתם את הגישה של הגרצה אותה התכוונתם להגיש. לא יתקבלו טענות על אי התאמאה בין הקובץ שנמצא ב-e-Moodle לבין הגרצה ש"התכוונתם" להגיש ולא יתקבלו הגשות מאוחרות במקרים הבאים.
- יש להגיש קובץ ארכיב מסוג Bzipped2-TAR בשם מהצורה (שרשור מספרי ת.ז – 9 ספרות):

```
proj-part2-<student1_id>-<student2_id>.tar.bz2
proj-part2-012345678-345678901.tar.bz2
```

לדוגמה:
- בארכיב יש לכלול את הקבצים הבאים:
 - את כל קבצי הקוד בהם השתמשתם (Bison, Flex, headers, Bison, וכו'). וכל קובץ קוד מקור הנדרש לבניית המנתח, כולל קבצי קוד מקור שספקו על ידי צוות הקורס).
 - makefile הבונה את המנתח התchieבי - שם קובץ הריצה של המנתח התchieבי הנוצר צריך להיות **part2**.
 - מסמך תיעוד קצר בפורמט PDF המכיל הסבר על התוכנית שלכם, מבני נתונים בהם השתמשתם והנחות שעשיתם.
- קובץ הארכיב צריך להיות "שטווח" (כלומר, שלא ייצור ספירות משנה בעת הפתיחה אלא הקבצים ייווצרו בספריה הנוכחית).
- חלק זה בפרויקט ינתן דגש רב על סדר ותיעוד בקוד. לפחות מבולגן ולא מתועד **ירדו נקודות!** כמו בחלק 1, סבבetta הבדיקה הרשמית הינה המבונה הויירטואלית של לינוקס המספקת לכם. ניתן לפתח במחשב אחר, אולם חובה عليكم לוודא שהתרגיל המוגש בנבנה ורץ היבט במבונה הויירטואלית הרשמית. לא יושרו דוחות בתרגיל לצורך התאמות למוכנות היעד הרשמי.
- **תרגיל שלא יצליח להתקmpl יקבל ציון 0.**

בהצלחה!