תורת הקומפילציה

תרגיל בית 2 – בניית מנתח תחבירי

מתרגלת אחראית: אביגיל ימפולסקי - avigaily@campus.technion.ac.il

ההגשה בזוגות

עבור כל שאלה על התרגיל, יש לעין ראשית **בפיאצה** ובמידה שלא פורסמה אותה השאלה, ניתן להוסיף אותה ולקבל מענה, אין לשלוח מיילים בנושא התרגיל בית כדי שנוכל לענות על השאלות שלכם ביעילות.

תיקונים לתרגיל יסומנו בצהוב, חובתכם להתעדכן בהם באמצעות קובץ התרגיל.

התרגיל ייבדק בבדיקה אוטומטית. **הקפידו למלא אחר ההוראות במדויק**. הבדיקה תתבצע על שרת הקורס csComp.

הנחיות כלליות

בתרגיל זה עליכם לממש ניתוח תחבירי לשפת FanC, הכוללת פעולות אריתמטיות, פונקציות, והמרות מובנות מ-byte (בית אחד) ל-int (4 בתים).

התוצר הסופי של המנתח יהיה AST אשר ישמש אתכם גם בתרגילים.

מנתח לקסיקלי

יש לכתוב מנתח לקסיקלי המתאים להגדרות הבאות:

אסימון	תבנית
VOID	Void
INT	Int
BYTE	byte
BOOL	bool
AND	and
OR	or
NOT	not
TRUE	true
FALSE	false
RETURN	return
IF	if
ELSE	else
WHILE	while
BREAK	break
CONTINUE	continue
SC	;
COMMA	,
LPAREN	(
RPAREN	)
LBRACE	{
RBRACE	}
ASSIGN	=

RELOP	== | != | < | > | <= | >=
BINOP	+ | - | * | /
ID	[a-zA-Z][a-zA-Z0-9]*
NUM	0 | [1-9][0-9]*
NUM_B	0b | [1-9][0-9]*b
STRING	"([^\n\r\"\\]|\\[rnt"\\])+"

ניתן לשנות את שמות האסימונים או להוסיף אסימונים נוספים במידת הצורך, כל עוד המנתח הלקסיקלי מזהה את כל התבניות לעיל.
יש להתעלם מרווחים, ירידות שורה משני הסוגים (LF ,CR) וטאבים כך שלא תתקבל עליהם שגיאה לקסיקלית.
יש להתעלם מהערות שורה (הערות C++) המיוצגות ע"י התבנית //[^\r\n]*[\r|\n|\r\n]?

הערה: *המנתח הזה שונה במקצת מהמנתח הלקסיקלי של תרגיל בית 1, אבל ניתן להשתמש במנתח שתופס אסימונים נוספים או מתמודד עם escape-ים נוספים, כמו המנתח של תרגיל בית 1, כל עוד תפסתם את כל התבניות בטבלה.*

תחביר

יש לכתוב מנתח תחבירי שיתאים לדקדוק הבא:

1. $Program \rightarrow Funcs$
2. $Funcs \rightarrow \epsilon$
3. $Funcs \rightarrow FuncDecl\ Funcs$
4. $FuncDecl \rightarrow RetType\ ID\ LPAREN\ Formals\ RPAREN\ LBRACE\ Statements\ RBRACE$
5. $RetType \rightarrow Type$
6. $RetType \rightarrow\ VOID$
7. $Formals \rightarrow \epsilon$
8. $Formals \rightarrow FormalsList$
9. $FormalsList \rightarrow FormalDecl$
10. $FormalsList \rightarrow FormalDecl\ COMMA\ FormalsList$
11. $FormalDecl \rightarrow Type\ ID$
12. $Statements \rightarrow Statement$
13. $Statements \rightarrow Statements\ Statement$
14. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
15. $Statement \rightarrow Type\ ID\ SC$
16. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
17. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
18. $Statement \rightarrow Call\ SC$
19. $Statement \rightarrow RETURN\ SC$
20. $Statement \rightarrow RETURN\ Exp\ SC$
21. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
22. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
23. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
24. $Statement \rightarrow BREAK\ SC$
25. $Statement \rightarrow CONTINUE\ SC$
26. $Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
27. $Call \rightarrow ID\ LPAREN\ RPAREN$

28. $ExpList \rightarrow Exp$
29. $ExpList \rightarrow Exp\ COMMA\ ExpList$
30. $Type \rightarrow INT$
31. $Type \rightarrow BYTE$
32. $Type \rightarrow BOOL$
33. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
34. $Exp \rightarrow Exp\ BINOP\ Exp$
35. $Exp \rightarrow ID$
36. $Exp \rightarrow Call$
37. $Exp \rightarrow NUM$
38. $Exp \rightarrow NUM\ B$
39. $Exp \rightarrow STRING$
40. $Exp \rightarrow TRUE$
41. $Exp \rightarrow FALSE$
42. $Exp \rightarrow NOT\ Exp$
43. $Exp \rightarrow Exp\ AND\ Exp$
44. $Exp \rightarrow Exp\ OR\ Exp$
45. $Exp \rightarrow Exp\ RELOP\ Exp$
46. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$

הערות:

1. הדקדוק כפי שמוצג כאן אינו חד משמעי ב-Bison. יש להפכו לחד משמעי תוך שימור השפה. בעיה לדוגמה שיש לפתור: http://en.wikipedia.org/wiki/Dangling_else.
יש לפתור את בעיית ה-Dangling else ללא שינוי הדקדוק אלא באמצעות מתן עדיפות לכללים או אסוציאטיביות מתאימה לאסימונים.

2. יש להקפיד על מתן עדיפויות ואסוציאטיביות מתאימים לאופרטורים השונים. יש להשתמש בטבלת העדיפויות כאן: http://introcs.cs.princeton.edu/java/11precedence

3. אין צורך לבצע שינויים בדקדוק, פרט לשם הבדלה בין האופרטורים השונים.

הוראות התרגיל

עליכם לכתוב הגדרת סכימת התרגום בשפת bison אשר תיצור AST של הקוד בערוץ הקלט הסטנדרטי.

מסופק לכם קובץ main.cpp אשר קורא לפונקציה yyparse() שנוצרת ע"י bison ועליה לעמוד בדרישות הבאות:

1. בסוף הניתוח שורש ה-AST תהיה שמור במשתנה גלובלי program. ההגדרת המשתנה מופיעה בשלד של **parser.y** המסופק לכם.
2. בעת זיהוי שגיאה לקסיקלית או תחבירית על מנתח לדווח על השגיאה מיד (כלומר בנקודה העמוקה ביותר בעץ הגזירה שבה ניתן לזהותה).

בקבצים **nodes.hpp/.cpp** מסופקים לכם הגדרות ומימושים של צמתים ב-AST תחת מחרב שמות ast.
כל הצמתים יורשים ממחלקת Node ולכן **YYSTYPE** מוגדר להיות מצביע חכם ל-Node, כלומר:

```
std::shared_ptr<ast::Node>
```

על כן, תוכלו לבצע השמה של הצמתים בעת יצירתם לתוך yylval (ב-flex) ו-$$/$* (ב-bison). למשל, על מנת ליצור צומת המתאים למספר שלם המיוצג על ידי לקסמה "23", מספיק לבצע השמה הבאה:

```
yylval = std::shared_ptr(new ast::Num("23"));
```

או בדרך מקוצרת ומאופטמת:

```
yylval = std::make_shared<ast::Num>("23");
```

במקרה ואתם צריכים להמיר מצביע חכם של Node לטיפוס מדויק יותר, תשתמשו בdynamic_pointer_cast.
למשל, על מנת להמיר משתנה node ל-Exp ניתן לכתוב ביטוי הבא:

```
std::dynamic_pointer_cast<ast::Exp>(node)
```

קלט ופלט המנתח

המנתח יקבל את הקלט מהערוץ הסטנדרטי לקלט (stdin). flex משתמש בערוץ הקלט הסטנדרתי כברירת מחדל ועליכם לקשר בין bison לבין flex.

הפונקציה main המסופקת לכם תקרא ל-yyparse ותריץ visitor מיוחד להדפסת AST שגם כן סופק לכם בקבצי output.hpp/cpp. ה-visitor מדפיס את העץ חל מהצומת שהוא קיבל ב-accept.

ניתן קבצי פלט לדוגמא. יש לבדוק כי המנתח שלכם פולט פלט זהה אליהם. הבדלים יגרמו לכישלון הבדיקות האוטומטיות.

טיפול בשגיאות

בקובץ הקלט יכולות להיות שגיאות לקסיקליות ותחביריות. **על המנתח לסיים את ריצתו מיד עם זיהוי שגיאה** (כלומר בנקודה העמוקה ביותר בעץ הגזירה שבה ניתן לזהותה).
ניתן להניח כי הקלט מכיל שגיאה אחת לכל היותר.

על מנת לדווח על שגיאות יש להשתמש בפונקציות הנתונות בקובץ **output.hpp**:

שגיאה לקסיקלית	errorLex(lineno)
שגיאה תחבירית	errorSyn(lineno)

בכל השגיאות הנ"ל lineno הוא מס' השורה בה מופיעה השגיאה.

- במקרה של שגיאה הפלט של המנתח יהיה דיווח על השגיאה בלבד.

הערות נוספות על התרגיל

- בתרגיל זה תדרשו להשלים קובץ **bison** יחיד בשם **parser.y**. שימרו עליו פשוט.
- לפני תחילת התרגיל תקראו את ההערות ב-nodes.hpp על מנת להבין אילו צמתים יש ב-AST ומה נדרש על מנת לבנות אותם.
- אתם יכולים, אך לא חייבים, לבצע שינויים ב-main.cpp, output.hpp/.cpp ו-nodes.hpp/.cpp.
- מומלץ לא להתעסק עם מצביעים רגילים, אלא להשתמש במצביעים חכמים, על מנת לא ליצור שגיאות זיכרון הקשות לדיבוג.
- מומלץ להתמש במחרוזות ++C (string) במקום מחרוזות C (const char*) איפה שניתן.

הוראות הגשה

מסופק לכם קובץ Makefile שאיתו תקומפל ההגשה שלכם. שימו לב כי קובץ ה-Makefile מאפשר שימוש ב-STL. אין לשנות את ה-Makefile.

יש להגיש קובץ אחד בשם ID1-ID2.zip, עם מספרי ת"ז של שתי המגישות. על הקובץ להכיל:

- קובץ flex בשם scanner.lex המכיל את כללי הניתוח הלקסיקלי.
- קובץ בשם parser.y המכיל את כללי הניתוח התחבירי.
- את כל הקבצים הנדרשים לבניית המנתח, כולל קבצים שסופקו כחלק מהתרגיל אם בחרתם להשתמש בהם.

בנוסף, יש להקפיד שהקובץ **לא יכיל** את:

- קובץ ההרצה.
- קבצי הפלט של flex ו-bison.
- קובץ Makefile שסופק כחלק מהתרגיל.

יש לוודא כי בביצוע unzip לא נוצרת תיקיה נפרדת. על המנתח להיבנות על השרת csComp ללא שגיאות באמצעות קובץ Makefile שסופק עם התרגיל. באתר הקורס מופיע קובץ zip המכיל קבצי בדיקה לדוגמה. יש לוודא כי פורמט הפלט זהה לפורמט הפלט של הדוגמאות הנתונות. כלומר, ביצוע הפקודות הבאות:

```
unzip id1-id2.zip
cp path-to/Makefile .
cp path-to/hw2-tests.zip  .
unzip hw2-tests.zip
make
./hw2 < t1.in 2>&1 > t1.res
diff t1.res path-to/t1.out
```

ייצור את קובץ ההרצה בתיקיה הנוכחית ללא שגיאות קומפילציה, יריץ אותו, ו-diff יחזיר 0.

הגשות שלא יעמדו בדרישות לעיל יקבלו ציון 0 ללא אפשרות לבדיקה חוזרת.

בדקו היטב שההגשה שלכן עומדת בדרישות הבסיסיות הללו לפני ההגשה עצמה.

שימו לב כי באתר מופיע script לבדיקה עצמית לפני ההגשה בשם selfcheck. תוכלו להשתמש בו על מנת לוודא כי ההגשה שלכם תקינה.

בתרגיל זה (כמו בתרגילים אחרים בקורס) **ייבדקו העתקות**. אנא כתבו את הקוד שלכם בעצמכם.

בהצלחה! ☺