תורת הקומפילציה

תרגיל בית 3 – בנית מנתח סמנטי

מתרגל אחראי: אנדריי בבין - andrey.babyn@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 תרגיל הבית 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.**

כללי

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

מנתח לקסיקלי

יש לכתוב מנתח לקסיקלי המתאים להגדרות הבאות:

אסימון	תבנית
INT	int
BYTE	byte
B	b
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	;
LPAREN	(
RPAREN	)
LBRACE	{
RBRACE	}
ASSIGN	=
RELOP	== | != | < | > | <= | >=
BINOP	+ | - | * | /
ID	[a-zA-Z][a-zA-Z0-9]*
NUM	0 | [1-9][0-9]*
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]?`

תחביר

יש לכתוב מנתח תחבירי שיתאים לדקדוק הבא:

1. $Program \rightarrow Statements$
2. $Statements \rightarrow Statement$
3. $Statements \rightarrow Statements\ Statement$
4. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
5. $Statement \rightarrow Type\ ID\ SC$
6. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
7. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
8. $Statement \rightarrow Call\ SC$
9. $Statement \rightarrow RETURN\ SC$
10. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
11. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
12. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
13. $Statement \rightarrow BREAK\ SC$
14. $Statement \rightarrow CONTINUE\ SC$
15. $Call \rightarrow ID\ LPAREN\ Exp\ RPAREN$
16. $Type \rightarrow INT$
17. $Type \rightarrow BYTE$
18. $Type \rightarrow BOOL$
19. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
20. $Exp \rightarrow Exp\ BINOP\ Exp$
21. $Exp \rightarrow ID$
22. $Exp \rightarrow Call$
23. $Exp \rightarrow NUM$
24. $Exp \rightarrow NUM\ B$
25. $Exp \rightarrow STRING$
26. $Exp \rightarrow TRUE$
27. $Exp \rightarrow FALSE$
28. $Exp \rightarrow NOT\ Exp$
29. $Exp \rightarrow Exp\ AND\ Exp$
30. $Exp \rightarrow Exp\ OR\ Exp$
31. $Exp \rightarrow Exp\ RELOP\ Exp$
32. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$

הערות:

1. הדקדוק כפי שמוצג כאן אינו חד משמעי ב-Bison. יש להפכו לחד משמעי תוך שימור השפה. בעיה לדוגמה שיש לפתור: https://en.wikipedia.org/wiki/Dangling_else.
   יש לפתור את בעיית ה-Dangling else (למשמעות כמו בשפת C) 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: https://introcs.cs.princeton.edu/java/11precedence
3. אין צורך לבצע שינויים בדקדוק, פרט לשם הבדלה בין האופרטורים השונים.

בדיקות סמנטיות

טבלאות סמלים

בשפת FanC קיים קינון סטטי של scopes: כל משתנה מוגדר ב-scope שבו הוכרז, ובכל הצאצאים של אותו scope. אסור להכריז על משתנה שכבר מוגדר באותו ה-scope (כלומר, הוכרז בscope הנוכחי או באבות שלו) – במילים אחרות, **אין shadowing של אף identifier שכבר מוגדר** (כולל identifier של פונקציה), ואסור להשתמש במשתנה או פונקציה שלא הוגדרו. שימוש במשתנה הוא כל מופע פרט להכרזה שלו. משתנה מוגדר החל מה-statement שאחרי הגדרתו.

קטעי הקוד הבאים תקינים תחבירית:

```
int a;
int a;
```

וגם:

```
int a;
c = 6;
```

אך לא נרצה לאפשר אותם בשפת FanC. לכן יש לנהל טבלאות סמלים.

בטבלת הסמלים נשמור עבור כל משתנה ופונקציה את שמו, מיקומו היחסי ברשומת ההפעלה, והטיפוס שלו.

יש להשתמש בטבלאות הסמלים כדי לבצע את הבדיקות הבאות:

1. בכל הכרזה על משתנה יש לוודא שמשתנה באותו שם לא מוגדר ב-scope הנוכחי או באחד ה-scopes המכילים אותו.
2. בכל שימוש במשתנה יש לוודא כי הוא מוגדר.
3. בכל שימוש בפונקציה, יש לוודא כי היא הוגדרה לפני השימוש. הפונקציות היחידות ב-FanC הן פונקציות ספריה.

קיימות שלוש פונקציות ספריה:

1. print אשר מקבלת מחרוזת (string) ומחזירה void.
2. printi אשר מקבלת int ומחזירה void.
3. readi אשר מקבלת int ומחזירה int.

יש להכניס את שלוש הפונקציות הנ"ל לטבלת הסמלים בפתיחת ה-scope הגלובלי בסדר הנ"ל, כלומר: קודם את print, לאחר מכן את printi ובסוף את readi.

בנוסף יש להשתמש בטבלת הסמלים כדי לבצע בדיקות טיפוסים לפי כללי הטיפוסים של FanC שמוגדרים בהמשך.

שימו לב כי בשביל לתמוך בפונקציות ייתכן שתצטרכו לשמור מידע נוסף פרט למידע לעיל.

כללי Scoping:

1. בלוק מייצר scope חדש.
2. if/else/while מייצרים scope חדש.

לכן, במקרה בו נפתח בלוק כחלק מפקודת if/else/while יפתחו שני scopes. אחד ריק עבור ה-if/while/else ואחד עבור הבלוק.

שימו לב כי כדי לשמור את print בטבלת הסמלים אנחנו מגדירים את string כטיפוס פנימי, למרות שהוא לא נגזר ע"י Type.

כללי טיפוסים

יש לקבוע את הטיפוסים של ביטויים לפי הכללים הבאים:

1. ביטוי שנגזר מ-NUM טיפוסו int, ומ-NUM B טיפוסו byte. לטיפוסים אלו נקרא הטיפוסים המספריים.
2. טיפוס הקבועים true/false הוא bool.
3. טיפוס קבוע מחרוזת הוא string.
4. הטיפוס של משתנה או קבוע נקבע לפי הגדרתו.
5. הטיפוס של ביטוי Call נקבע לפי טיפוס ההחזרה של הפונקציה הנקראת.
6. ניתן לבצע השמה של ביטוי מטיפוס מסוים למשתנה מאותו הטיפוס.
7. ניתן לבצע השמה של byte ל-int.
8. ניתן לבצע השמה מפורשת מint או מbyte לint או לbyte באמצעות הפקודה (byte)<value> או (int)<value> כאשר value הוא ביטוי מטיפוס int או byte.
9. פעולות relop מקבלות שני אופרנדים מטיפוסים מספריים. טיפוס ההחזרה של הביטוי הוא bool.
10. פעולות לוגיות (and, or, not) מקבלות אופרנדים מטיפוס bool. טיפוס ההחזרה של הביטוי הוא bool.
11. פעולות binop מקבלות שני אופרנדים מספריים. טיפוס החזרה של binop הוא הטיפוס עם טווח הייצוג הגדול יותר מבין שני הטיפוסים של האופרנדים.
12. ביטוי מסוג string ניתן לשימוש רק בקריאה לפונקציית הספרייה print.
13. פונקציית הספריה print מקבלת ארגומנט אחד מסוג string ומחזירה void.
14. פונקציית הספריה printi מקבלת ארגומנט אחד מסוג int (או byte) ומחזירה void.
15. פונקציית הספריה readi מקבלת ארגומנט אחד מסוג int (או byte) ומחזירה int.
16. ניתן לקרוא לפונקציה בהעברת פרמטר תואם לטיפוס בהגדרת הפונקציה. מותר להעביר ביטוי e_i לפרמטר p_i של הפונקציה אם השמה של e_i למשתנה המוגדר מהטיפוס של p_i מותרת.
17. פקודות if ו-while מקבלות Exp מטיפוס בוליאני.

שימו לב! בכל מקרה שלא מוגדר בכללים אלה יש להחזיר שגיאה. ראו סעיף "טיפול בשגיאות" בהמשך.

בדיקות סמנטיות נוספות

בנוסף, יש לבצע את הבדיקות הבאות, שאינן בדיקות טיפוסים:

1. עבור כלל הדקדוק $Statement \rightarrow BREAK\ SC$ ועבור הכלל $Statement \rightarrow CONTINUE\ SC$ יש לבצע בדיקה כי הם מתגלים רק בתוך לולאת while, אחרת יש לעצור עם שגיאת UnexpectedBreak או UnexpectedContinue בהתאמה ולצאת מהתכנית.
2. ליטרל שטיפוסו byte לא יציין מספר הגדול מ-255.

שימו לב שאין חובה שהתוכנית תכיל פקודת return.

מיקום המשתנים בזיכרון

בתרגיל אנו מניחים שכל משתנה הוא בגודל 1, ללא תלות בטיפוס. אזי עבור הקוד הבא:

```
int x;
{
      bool y;

      byte z;
}
bool w;
```

המיקומים (offset) לכל משתנה יהיו:

x	0
y	1
Z	2
w	1

קלט ופלט המנתח

קובץ ההרצה של המנתח יקבל את הקלט מ-stdin.

יש להיעזר בקובץ hw3_output.hpp המצורף לתרגיל על מנת לייצר פלט הניתן לבדיקה אוטומטית.

בסוף כל scope, כולל ה- scope הגלובאלי, המנתח ידפיס את המשתנים שהוגדרו ב- scope זה ל-stdout **בסדר הבא**:

1. קריאה לפונקציה endScope
2. עבור כל identifier שהוגדר ב-scope על פי סדר ההכרזה יש לקרוא לפונקציה printID(id,offset,type) עם שם המשתנה/פונקציה, המיקום בזיכרון, והטיפוס.
   a. עבור כל משתנה מחרוזת הטיפוס צריכה להיות זהה לשם האסימון שהוגדר לטיפוס בחלק הלקסיקלי בתיאור התרגיל (עבור מחרוזת, הטיפוס הוא STRING).
   b. עבור פונקציה, כדי לקבל את מחרוזת ה-type יש לקרוא לפונקציה makeFunctionType עם טיפוס הפרמטר וטיפוס ההחזרה כפי שהוגדרו בפרק "טבלאות סמלים". בנוסף, המיקום בזיכרון של פונקציה הוא תמיד 0.
3. שימו לב לבצע זאת בסוף כל scope לפי ההגדרה בפרק טבלת הסמלים של תיאור התרגיל.

ניתן קובץ פלט לדוגמא. יש לבדוק שהפורמט שהודפס זהה אליו. הבדלי פורמט יגרמו לכישלון הבדיקות האוטומטיות.

טיפול בשגיאות

בקובץ הקלט יכולות להיות שגיאות לקסיקליות, תחביריות וסמנטיות. **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)
שימוש במשתנה שלא הוגדר או ב-identifier שאינו משתנה כמשתנה (הכרזה אינה נחשבת כשימוש ב-identifier)	errorUndef(lineno, id)
שימוש בפונקציה שלא הוגדרה או ב-identifier שאינו פונקציה כפונקציה	errorUndefFunc(lineno, id)
ניסיון להגדיר identifier שכבר הוגדר	errorDef(lineno, id)
ניסיון להשתמש בפונקציה עם ארגומנט לא תואם. type יהיה הטיפוס המצופה.	errorPrototypeMismatch(lineno, id, type)
אי התאמה של טיפוסים (פרט להעברת פרמטר לא תואם לפונקציה)	errorMismatch(lineno)
פקודת break שאינה חלק מלולאה	errorUnexpectedBreak(lineno)
פקודת continue שאינה חלק מלולאה	errorUnexpectedContinue (lineno)
ליטרל מסוג byte מכיל מספר גדול מדי, כאשר value הוא הערך הקיים בקוד.	errorByteTooLarge(lineno, value)

בכל השגיאות הנ"ל id הוא שם המשתנה או הפונקציה, ו-lineno הוא מס' השורה בה מופיעה השגיאה.

- במקרה של שגיאה הפלט של המנתח יהיה תוכן כל ה-scopes שנעשה להם reduce והשגיאה שהתגלתה (כפי שניתן לראות בדוגמה t2). ניתן להניח שאחרי סיום scope תמיד יופיע לפחות אסימון (או תוSS סיום פלט) אחד תקין ומתאים לפי דקדוק ומצב האוטמט.
- יש לתפוס את השגיאה ולעצור את המנתח מוקדם ככל הניתן. לדוגמה, במקרה שבתנאי if מופיע Exp שאינו מטיפוס בוליאני, יש לזרוק את השגיאה ולעצור לפני ההדפסה שמתבצעת בסוף הscope.

והערות

סדר מומלץ לביצוע התרגיל (מומלץ להריץ בדיקות לאחר כל סעיף):

1. כתבו מנתח לקסיקלי ותחבירי ללא כללים סמנטיים.
2. בדקו שהמבנה התחבירי של השפה נאכף ואין אף קונפליקט.
3. הגדירו את YYSTYPE וממשו טבלאות סמלים. השתדלו ליצור מחלקות לכל נונטרמינל ולא ליצור struct אחד שמכיל את כל התכונות הסמנטיות.
מלבד הצורה שראיתם בתרגול לעשות זאת, ניתן לעשות זאת גם ע״י הגדרת union המכיל את כל ה-structs או הטיפוסים האפשריים. והגדרת כל טרמינל ונונטרמינל כ-struct או טיפוס המתאים לו.
להסבר ולדוגמה פשוטה עבור דקדוק המכיל טרמינלים NUM ו-OP ונונטרמינל exp נוסיף בחלק ה-declarations:

```
%union {
int val;
char op;
};
%token <val> NUM
%token <op> OP
%type  <val> exp
```

4. מומלץ מאוד לממש מחלקות לטיפול בדרישות שונות ולהפנות אליהן מהקוד בקובץ הדקדוק. שימוש בקוד חיצוני יחסוך לכם להריץ את bison בכל שינוי של הקוד. שימו לב כי ניתן להגיש קבצי קוד נוספים.
5. בצעו בדיקות סמנטיות.

שימו לב כי התרגיל לא ייבדק עם הכלי valgrind. על אף זאת, על התרגיל לא לקרוס. לכם כמובן מותר לבדוק עם valgrind או כל כלי אחר.

הוראות הגשה

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.ypp המכיל את המנתח
- את כל הקבצים הנדרשים לבניית המנתח, כולל *.hw3_output שסופקו כחלק מהתרגיל.

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-hw3 .
cp path-to/hw3-tests.zip  .

unzip hw3-tests.zip
make -f Makefile-hw3
./hw3 < 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-hw3. 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! ☺