

תורת הקומפילציה

תרגיל 3

מתרגל אחראי: יונתן יעקבי – yonatan.yac@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.**

כללי

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

מנתח לקסיקלי

יש לכתוב מנתח לקסיקלי המתאים להגדרות הבאות:

אסימון	תבנית
VOID	void
INT	int
BYTE	byte
B	b
BOOL	bool
CONST	const
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	}
LBRACKET	[
RBRACKET]
ASSIGN	=
RELOP	== != < > <= >= in
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, 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. $FormalDecl \rightarrow TypeAnnotation Type ID$
12. $Statements \rightarrow Statement$
13. $Statements \rightarrow Statements Statement$
14. $Statement \rightarrow LBRACE Statements RBRACE$
15. $Statement \rightarrow TypeAnnotation Type ID SC$
16. $Statement \rightarrow TypeAnnotation Type ID ASSIGN Exp SC$
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. $TypeAnnotation \rightarrow \epsilon$
34. $TypeAnnotation \rightarrow CONST$
35. $Exp \rightarrow LPAREN Exp RPAREN$
36. $Exp \rightarrow Exp BINOP Exp$
37. $Exp \rightarrow ID$
38. $Exp \rightarrow Call$
39. $Exp \rightarrow NUM$
40. $Exp \rightarrow NUM B$
41. $Exp \rightarrow STRING$
42. $Exp \rightarrow TRUE$
43. $Exp \rightarrow FALSE$

44. $Exp \rightarrow NOT\ Exp$
45. $Exp \rightarrow Exp\ AND\ Exp$
46. $Exp \rightarrow Exp\ OR\ Exp$
47. $Exp \rightarrow Exp\ RELOP\ Exp$
48. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$

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 (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: <http://introcs.cs.princeton.edu/java/11precedence>
3. אין צורך לבצע שינויים בדקדוק, פרט לשם הבדלה בין האופרטורים השונים.
4. שימו לב לשינויים מתרגילי בית קודמים.

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, טיפוס או ערך שכבר מוגדר באותו ה-scope – כלומר אין shadowing של אף identifier שכבר מוגדר (כולל identifier של פונקציה), ואסור להשתמש במשתנה, פונקציה, טיפוס או ערך 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, פרמטר 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. כלומר: מותר לקרוא לכל פונקציה שהוגדרה לפני הפונקציה הנוכחית, ומותר לקרוא לפונקציה הנוכחית (רקורסיה).

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. פונקציה ובלוק מייצרים scope חדש. פרמטרים של פונקציה שייכים ל-scope של הפונקציה.
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.

בנוסף קיימות שתי פונקציות ספריה: `print` ו-`printi`, כאשר `print` מקבלת מחרוזת (string) ו-`printi` מקבלת `int`. שתיהן מחזירות `void`. יש להכניס את שתי הפונקציות הנ"ל לטבלת הסמלים בפתיחת הscope הגלובלי בסדר הבא: קודם את `print` ולאחר מכן את `printi`.

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-`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 של `int` ל-`byte` או להפך באמצעות הפקודה `<value>(byte)` או `<value>(int)`.
9. ניתן לבצע השמה יחידה לטיפוס שהוא `const`, וזה בעת הגדרתו. בנוסף, לא ניתן להגדיר טיפוס `const` בלי לתת לו ערך, מלבד פרמטרים לפונקציה שערכם יקבע בעת הקריאה לפונקציה.
10. פעולות `relop` מקבלות שני אופרנדים מטיפוסים מספריים. טיפוס ההחזרה של הביטוי הוא `bool`.
11. פעולות לוגיות (`and`, `or`, `not`) מקבלות אופרנדים מטיפוס `bool`. טיפוס ההחזרה של הביטוי הוא `bool`.
12. פעולות `binop` מקבלות שני אופרנדים מספריים. טיפוס החזרה של `binop` הוא הטיפוס עם טווח הייצוג הגדול יותר מבין שני הטיפוסים של האופרנדים.
13. ביטוי מסוג `string` ניתן לשימוש רק בקריאה לפונקציית הספרייה `print`.
14. פונקציית הספרייה `print` מקבלת ארגומנט אחד מסוג `string` ומחזירה `void`.
15. פונקציית הספרייה `printi` מקבלת ארגומנט אחד מסוג `int` או `byte` ומחזירה `void`.
16. ניתן לקרוא לפונקציה בהעברת מספר נכון של פרמטרים תואמים לטיפוסים בהגדרת הפונקציה (לפי הסדר). מותר להעביר ביטוי e_i לפרמטר p_i של הפונקציה אם השמה של e_i למשתנה המוגדר מהטיפוס של p_i מותרת.
17. באותו אופן, בפונקציה המחזירה ערך, טיפוס ה-`Exp` בכל `RETURN Exp` חייב להיות מותר להשמה לטיפוס ההחזרה בהגדרת הפונקציה.
18. 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 → BREAK SC` ועבור הכלל `Statement → CONTINUE SC` יש לבצע בדיקה כי הם מתגלים רק בתוך לולאת `while`, אחרת יש לעצור עם שגיאת `UnexpectedBreak` או `UnexpectedContinue` בהתאמה ולצאת מהתכנית.
2. עבור כללי הדקדוק `Statement → RETURN SC` ו-`Statement → RETURN Exp SC` יש לבצע בדיקה כי הם תואמים לטיפוס הפונקציה: `RETURN Exp SC` מותר לשימוש רק בפונקציות שלא מחזירות `void` (בדיקת הטיפוס עבורו מפורטת תחת "כללי טיפוסים"), ו-`RETURN SC` רק בפונקציה המחזירה `void`. אחרת יש לעצור עם שגיאת `Mismatch` ולצאת מהתכנית.
3. **שימו לב** שאין חובה שפונקציה תכיל פקודת `return` ואין צורך לבדוק שלפונקציה המחזירה ערך קיימת פקודת `return`.
4. ליטרל שטיפוסו `byte` לא יציין מספר גדול מ-255.
4. קיימת בדיוק פונקציית `main` אחת, ללא פרמטרים, ועם טיפוס החזרה `void`.

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:

0	x
1	y
2	z
1	w

בנוסף, נמקם ארגומנטים של פונקציה בסדר הפוך ברשומת ההפעלה לפני מיקום 0. לכן עבור הפונקציה הבאה:

```
bool isPassing(int grade, int factor)
{
    return (grade+factor) > 55;
}
```

המיקומים יהיו:

-1	grade
-2	factor

קלט ופלט המנתח

קובץ ההרצה של המנתח יקבל את הקלט מ-stdin.

יש להיעזר בקובץ 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 בקוד (במידה ומדובר ב-scope של פונקציה, יש להתחיל מהפרמטרים, לפי סדר הגדרתם) יש לקרוא לפונקציה `printID(id,offset,type)` עם שם המשתנה, המיקום בזיכרון, והטיפוס.
 - a. עבור משתנה, קבוע או פרמטר, 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 הפרמטרים וטיפוס ההחזרה כפי שהוגדרו בסעיף הקודם. 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:

errorLex(lineno)	שגיאה לקסיקלית
errorSyn(lineno)	שגיאה תחבירית
errorUndef(lineno, id)	שימוש במשתנה שלא הוגדר או ב-identifier שאינו משתנה כמשתנה

errorUndefFunc(lineno, id)	שימוש בפונקציה שלא הוגדרה או ב-identifier שאינו פונקציה כפונקציה
errorDef(lineno, id)	ניסיון להגדיר identifier שכבר הוגדר
errorPrototypeMismatch(lineno, id, types)	ניסיון להשתמש בפונקציה עם ארגומנטים לא תואמים. types יהיה רשימת הטיפוסים המצופים.
errorMismatch(lineno)	אי התאמה של טיפוסים (פרט להעברת פרמטרים לא תואמים לפונקציה)
errorUnexpectedBreak(lineno)	פקודת break שאינה חלק מלולאה
errorUnexpectedContinue(lineno)	פקודת continue שאינה חלק מלולאה
errorMainMissing()	לא מוגדרת פונקציית void main()
errorByteTooLarge(lineno, value)	ליטרל מסוג byte מכיל מספר גדול מדי, כאשר value הוא הערך הקיים בקוד.
errorConstDef(lineno)	הגדרה של const בלי ערך
errorConstMismatch(lineno)	השמה לconst שהוגדר בעבר

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).
- 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 scope.
- בדיקה כי קיימת פונקציית void main() תתבצע לפני reduce של ה-scope הגלובלי. ולכן על המנתח לזהות זאת לפני הדפסת תוכן ה-scope הגלובלי.

הדרכה 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 נוסף 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.

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

unzip hw3-tests.zip
make
./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. תוכלו להשתמש בו על מנת לוודא כי ההגשה שלכם תקינה.

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

בהצלחה! 😊