תורת הקומפילציה

תרגיל בית 3 – בניית מנתח סמנטי

מתרגלת אחראית: ליאן מח'ול – layanmakhoul@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, חובתכן להתעדכן בהם באמצעות קובץ התרגיל.

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

הנחיות כלליות

בתרגיל בית הקודם בניתן מנתח תחבירי לשפת FanC, אשר יצר AST עבור קוד מקור בערוץ הקלט הסטנדרטי.

בתרגיל זה עליכן לממש ניתוח סמנטי לשפת FanC, על ידי מעבר על ה-AST והפעלת כללים סמנטים.

הוראות התרגיל

עליכן לכתוב מנתח סמנטי לפי שיטת Visitor אשר נלמדה בכיתה.

צמתי AST אשר סופקו לכן בתרגיל בית הקודם כבר מכילים את ההגדרה של מטודה accept. תצרו מחלקה חדשה אשר יורשת מ-Visitor וממשת מטודה visit עבור כל סוג צומת בעץ, בדומה ל-PrintVisitor מתרגיל בית הקודם.

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 או פונקציה שכבר מוגדר באותו ה-scope (או באב קדמון של אותו scope) – כלומר **אין הסתרה (shadowing) של אף מזהה (identifier) שכבר מוגדר**. כמו כן, אסור להשתמש במשתנה או פונקציה שלא הוגדרו. שימוש במשתנה הוא כל מופע פרט להכרזה שלו. משתנה מוגדר החל מהמשפט (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 לפחות את שמו, מיקומו היחסי ברשומת ההפעלה, וטיפוסו.

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 מוגדרת **במקום כלשהו בקוד**. כלומר, ניתן לקרוא לפונקציה גם לפני הגדרתה, כל עוד ההגדרה קיימת בקוד מקור (או לפני השימוש, או אחריו).

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

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

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 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 מ-int או מ-byte ל-int או ל-byte באמצעות הביטוי (byte)<value> או (int)<value> כאשר value הוא ביטוי מטיפוס מספרי.
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 (טווח ייצוג של int גדול יותר מטווח ייצוג של byte).
12. ביטוי מסוג string ניתן לשימוש רק בקריאה לפונקציית הספרייה print.
13. פונקציית הספריה print מקבלת ארגומנט אחד מסוג string ומחזירה void.
14. פונקציית הספרייה printi מקבלת ארגומנט אחד מסוג int או byte ומחזירה void.
15. 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.

16. באותו אופן, בפונקציה המחזירה ערך, טיפוס ה-Exp בכל $RETURN\ Exp$ חייב להיות מותר להשמה לטיפוס ההחזרה בהגדרת הפונקציה.
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 פקודות Break ו-Continue יש לבדוק כי הם מתגלים רק בתוך לולאת while, אחרת יש לעצור עם שגיאת UnexpectedBreak או UnexpectedContinue בהתאמה.
2. עבור כללי הדקדוק $Statement \rightarrow RETURN\ SC$ ו-$Statement \rightarrow RETURN\ Exp\ SC$ 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.
   שימו לב שאין חובה שפונקציה תכיל פקודת return ואין צורך לבדוק שלפונקציה המחזירה ערך קיימת פקודת return.
3. **ליטרל** שטיפוסו byte לא יציין מספר הגדול מ-255, אחרת יש לעצור עם שגיאת ByteTooLarge.
4. קיימת בדיוק פונקציית main אחת, ללא פרמטרים, ועם טיפוס החזרה void. אחרת, יש לעצור עם שגיאה MainMissing.

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

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:

grade	-1
factor	-2

קלט ופלט המנתח

המנתח הסמנטי יקבל את הקלט מהערוץ הסטנדרטי לקלט (stdin). המנתח התחבירי מתרגיל בית הקודם מקבל קלט מערוץ הסטנדרטי ולכן, עליכן להעביר את ה-AST למנתח הסמנטי.

קבצי **output.hpp/.cpp** המצורפים לתרגיל הורחבו לאחר תרגיל הבית הקודם ומכילים פונקציות חדשות לדיווח על שגיאות ומנגנון יצירת פלט הניתן לבדיקה אוטומטית.

עליכן ליצור משתנה מטיפוס ScopePrinter ולהשתמש בו באופן הבא:

1. בתחילת כל scope, פרט ל-scope הגלובלי, יש לקרוא למטודה beginScope.
2. בסוף כל scope, פרט ל-scope הגלובלי, יש לקרוא למטודה endScope.
3. עבור כל משתנה אשר הוגדר ב-scope, על פי סדר ההכרזה בקוד (במידה ומדובר ב-scope של פונקציה, יש להתחיל מהפרמטרים, לפי סדר הגדרתם) יש לקרוא למטודה emitVar(id, type, offset) עם שם המשתנה, הטיפוס והמיקום בזיכרון. הקריאות אלו צריכים להתבצע בין beginScope ו-endScope הרלוונטים.
4. עבור כל פונקציה, על פי סדר ההכרזה בקוד, יש לקרוא למטודה emitFunc(id, returnType, paramTypes) עם שם הפונקציה, טיפוס החזרתה וטיפוס הפרמטרים שלה. ניתן לקרוא למטודה זו בכל עת, ללא תלות ב-beginScope ו-endScope.
5. בסיום מוצלח של ניתוח סמנטי יש להעביר את המשתנה של ScopePrinter לערוץ הפלט הסטנדרטי. ראו דוגמה למטה.

שימו לב לבצע זאת בסוף כל scope לפי ההגדרה בפרק טבלאות סמלים.

למשל, עבור קוד מקור הבא:

```
void main() {
      int a;
      if (a == 3) int c = 2;
      foo(a);
}
void foo(int p) {}
```

שימוש תקין ב-ScopePrinter:

```
ScopePrinter printer;
// emit library functions
printer.emitFunc("print", BuiltInType::VOID, {BuiltInType::STRING});
printer.emitFunc("printi", BuiltInType::VOID,
{BuiltInType::STRING});
// emit main function defined in the code
printer.emitFunc("main", BuiltInType::VOID, {});
printer.emitFunc("foo", BuiltInType::VOID, {BuiltInType::INT});


printer.beginScope(); // main function scope
printer.emitVar("a", BuiltInType::INT, 0); // variable a defined
```

```
printer.beginScope(); // if scope
printer.emitVar("c", BulitInType::INT, 1); // variable c defined;
printer.endScope(); // if scope
printer.endScope(); // main function scope
printer.beginScope(); // foo function scope
printer.emitVar("p", BuiltInType::INT, -1); // function's parameter
printer.endScope(); // foo function scope
std::cout << printer; // print the output
```

שימו לב כי הקריאות ל-emitFunc ניתן לבצע לא רק בהתחלה, אלא בכל עת, אך צריך להקפיד על הסדר בין emitFunc שונים.

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 במזהה בתור משתנה כאשר המזהה אינו מוגדר.	errorUndef(lineno, id)
שימוש במזהה של פונקציה בתור משתנה	errorDefAsFunc(lineno, id)
שימוש במזהה בתור פונקציה כאשר המזהה אינו מוגדר.	errorUndefFunc(lineno, id)
שימוש במזהה של משתנה בתור פונקציה	errorDefAsVar(lineno, id)
ניסיון להגדיר identifier שכבר הוגדר	errorDef(lineno, id)
ניסיון להשתמש בפונקציה עם ארגומנטים לא תואמים. types יהיה רשימת הטיפוסים **המצופים**.	errorPrototypeMismatch(lineno, id, types)
אי התאמה של טיפוסים (פרט להעברת פרמטרים לא תואמים לפונקציה)	errorMismatch(lineno)
פקודת break שאינה חלק מלולאה	errorUnexpectedBreak(lineno)
פקודת continue שאינה חלק מלולאה	errorUnexpectedContinue (lineno)
לא מוגדרת פונקציית void main()	errorMainMissing()
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.
שימו לב כי ב-Node קיים שדה של lineno אשר מכיל שורה בה הצומת הוגדר.

במקרה של שגיאה הפלט של המנתח תהיה הודעת שגיאה בלבד.

הערות נוספות על התרגיל

- מומלץ להסתכל על המימוש של PrintVisitor בתור דוגמה לפני הבנייה של ה-Visitor שלכם.
- שימו לב כי בשיטת Visitor אתם שולטים על סדר מעבר על AST על ידי קריאות ל-accept.
- מומלץ ליצור ScopePrinter בתור שדה ב-Visitor שלכם.
- מומלץ להוסיף שדות חדשים לצמתים שונים בתור תכונות סמנטיות. חשבו היטב אילו תכונות סמנטיות אתן רוצות להוסיף ולאילו סוגים של צמתים.
- שימו לב כי ניתן לקרוא לפונקציות לפני הגדרתם והדבר יכול לדרוש מכם טיפול מיוחד.

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

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/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.

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

בהצלחה! ☺