21.02.2021

מבחן סוף סמסטר – מועד א׳

מרצה אחראי: ד״ר שחר יצחקי

מתרגלים: מתן פלד, איתן סינגר, עומר בלחסין

הוראות:

א. בטופס המבחן 13 עמודים מהם 6 דפי נוסחאות. בדקו שכל העמודים ברשותכם.

ב. משך המבחן **שעתיים וחצי** (150 **דקות).**

ג. אסור כל חומר עזר חיצוני.

ד. במבחן 4 שאלות. כל השאלות הינן חובה. משקל כל שאלה מופיע בראשיתה. (חלוקת המשקל בין הסעיפים בכל שאלה אינה בהכרח אחידה.)

ה. ניתן לציין לגבי סעיף או שאלה ״לא יודע/ת״. תשובה זו תזכה ב- 20% מהניקוד של הסעיף או השאלה. תשובות שגויות לא יזכו בניקוד.

ו. חובה לנמק כל תשובה. לא יינתן ניקוד על תשובות ללא נימוק.

ז. קראו את כל המבחן לפני שאתם מתחילים לענות על השאלות.

ח. אין צורך להגיש את הטופס בתום הבחינה.

ט. את התשובות לשאלות יש לרשום במחברת הבחינה בלבד.

בהצלחה!

שאלה 1: שלבי הקומפילציה ואופטימיזציות (20 נק׳)

חלק א׳: סיווג מאורעות (10 נק׳)

נתונה התוכנית הבאה בשפת FanC:

```
1  int foo(int n) {
2     int c = 1;
3     while (c != 0) {
4        c = c - 1;
5        if (n > 100) {
6           n = n - 10;
7        } else {
8           n = n + 11;
9           c = c + 2;
10       }
11    }
12    return n;
13 }
14 int bar(int x) {
15    x = x * 2 + 2;
16    if (x > 50) {
17       return x - 50;
18    } else {
19       return x + 50;
20    }
21 }
22 void main() {
23    int acc = 0;
24    int i = 0;
25    while (i < 500) {
26       acc += foo(bar(42));
27    }
28 }
```

Syntax error lol

בסעיפים הבאים מוצגים שינויים (בלתי תלויים) לקוד של התוכנית, או מאורע שמתרחש בזמן השימוש בקוד.

שלבי הקומפילציה הם: ניתוח לקסיקלי, ניתוח תחבירי, ניתוח סמנטי, ייצור קוד, זמן ריצה.

עבור כל שינוי כתבו האם הוא גורם לשגיאה, ואם כן, ציינו את השלב המוקדם ביותר שבה נגלה אותה ונמקו בקצרה:

int to byte conversion

semantic error b.c of line 26

semantic error 26

lex error '@'

א. (2 נק׳) מחליפים את שורה 14 בשורה הבאה: byte bar(byte x)

ב. (2 נק׳) מחליפים את שורה 1 בשורה הבאה: int foo(byte n)

ג. (2 נק׳) מחליפים את שורה 12 בשורה הבאה: return @n;

עבור כל מאורע, ציינו את **כל** שלבי הקומפילציה בהם הוא יכול להתרחש ונמקו בקצרה.

ד. (2 נק׳) נוצר קוד מכונה עבור הפונקציה bar. אין מקום לזיכרון

ה. (2 נק׳) בשורה 15, **נקבע** האם תתבצע קודם פעולת הכפל או פעולת החיבור (סדר פעולות חשבון). syntax (לא בוי)

<u>חלק ב׳ : אופטימיזציות (10 נק׳)</u>

נניח שקיים קומפיילר עבור שפת FanC שמשתמש בטכניקות JIT, כפי שנלמדו בכיתה.

א. (5 נק׳) פרטו אילו אופטימיזציות ניתן לבצע באמצעות JIT בתוכנית הנתונה בחלק א׳ **שלא ניתן** לבצע ללא JIT (במסגרת האופטימיזציות שנלמדו). נק׳ אם יודע שצריך לפעמים (Interpreter) ובשאר פעמים הרצה מהירה כפל ל bar לאחר

ב. (5 נק׳) אחרי אופטימיזציות JIT אם מצאתם כאלה שפעלו על bar, בצעו עליה את האופטימיזציות הבאות (אלה שרלוונטיות) :

acc += foo(bar(42)) פעולה בלתי

acc = const

1. Constant propagation
2. Copy propagation
3. Constant folding
4. Common sub-expression elimination
5. Algebraic simplification
6. החלפת קפיצה מותנית בקפיצה לא-מותנית
7. Branch chaining
8. Useless code elimination

רשמו את מספרי השורות שבהן ביצעתם את השינויים, בתוספת השינוי ואת סוג האופטימיזציה שגרמה לכך.

שאלה 2: אנליזה סטטית (30 נק׳)

מורטי כותב בשפת תכנות שבה יש ניהול זיכרון דינמי של אובייקטים על-ידי משפטים מהצורה הבאה:

```
x := new A
delete x
```

ובנוסף משפטי השמה רגילים וגישות לשדות של אובייקטים כנהוג בתכנות מונחה-עצמים.

כדי שלא להתבלבל, מורטי **נמנע לחלוטין מ-**aliasing בין משתנים מטיפוס מצביע, כלומר מהשמות מהצורה `x := y` כאשר x, y הם מצביעים (ובכלל זה שדות מטיפוס מצביע).

כמו-כן בשפה אין casting בין טיפוסים.

א. (15 נק׳) מורטי מודאג מכך שעקב ניהול הזכרון הידני בתכניות שלו הוא עלול לטעות ולשחרר את אותו אובייקט פעמיים, מה שכמובן יגרום לקריסת התוכנית והיקום.

הציעו למורטי אנליזה אשר בודקת קיומו של מצב זה, כלומר מדווחת על שגיאה במקרה של חשד לשחרור כפול. האנליזה חייבת להיות **נאותה**, כלומר: אם היא אינה מדווחת על שגיאה, אז **בוודאות** אין בתכנית שחרור כפול. הציעו אנליזה מדויקת ככל האפשר כדי לא להלחיץ את מורטי סתם עם התראות שווא.
האנליזה צריכה לבדוק כל פונקציה בנפרד, ובמקרה של משפטים המכילים קריאה לפונקציה אחרת, הניחו שאפשר להתעלם מהקצאות/שחרור זכרון שנעשות על-ידי הפונקציה הנקראת; כלומר, אפשר להניח שכל פונקציה משחררת את הזכרון שהיא הקצתה ורק אותו.

יש להתייחס להיבטים הבאים:

- הסריג שישמש כתחום הערכים (domain) עבור האנליזה, כולל יחס הסדר ופעולת join.
- פונקציות המעברים המתאימות עבור משפטים בשפה (ניתן להניח שהשפה כוללת, בנוסף להקצאה ושחרור, גם משפטי השמה ותנאים בדומה ל-FanC). זכרו שהפונקציות חייבות להיות **מונוטוניות**.
- האופן שבו הקומפיילר **ישתמש בתוצאות האנליזה** כדי להפיק את הודעות השגיאה הרצויות.

ב. (15 נק׳) ריק מגיע לגראז׳ ואומר למורטי: ״אתה פתי חסר-שכל. האנליזה שנתנו לך עובדת רק בגלל סגנון התכנות הנחות שלך שבו אין aliasing. אם היית מספיק חכם לכתוב השמות כמו `x := y`, האנליזה לא הייתה נאותה.״

האם ריק צודק? הוכיחו או הפריכו בעזרת הסבר/דוגמה.

שאלה 3: ניתוח תחבירי (20 נק׳)

נוסחאות מצורת DNF מוגדרות כאוסף של פסוקיות (conjunctions) המחוברות ביניהן על ידי פעולת ״או״ (סימן V) כאשר כל פסוקית בנויה מאוסף של ליטרלים (משתנים ושלילות משתנים) המחוברים ביניהם על ידי פעולת ״וגם״ (סימן ∧).

נתון הדקדוק הבא שמגדיר תת קבוצה מסוימת של נוסחאות DNF:

$S' \to D$

1. DNF2 → CS $
2. CS → C <u>v</u> CS
3. CS → C
4. C → L <u>∧</u> L
5. L → <u>var</u>
6. L → <u>¬</u> <u>var</u>

$CS \vee C$

$S' \to \cdot D$
$D \to \cdot CS$
$CS \to \cdot CS \vee C$
$CS \to \cdot C$
$C \to \cdot L \wedge L$
$L \to \cdot var$
$L \to \cdot \neg var$

$\xrightarrow{CS}$

הטרמינלים בדקדוק מסומנים בקו תחתון.

select(DNF2 → CS $) = first(CS $) = {var, ¬}

select(CS → C v CS) = {var, ¬}
select(CS → C) =

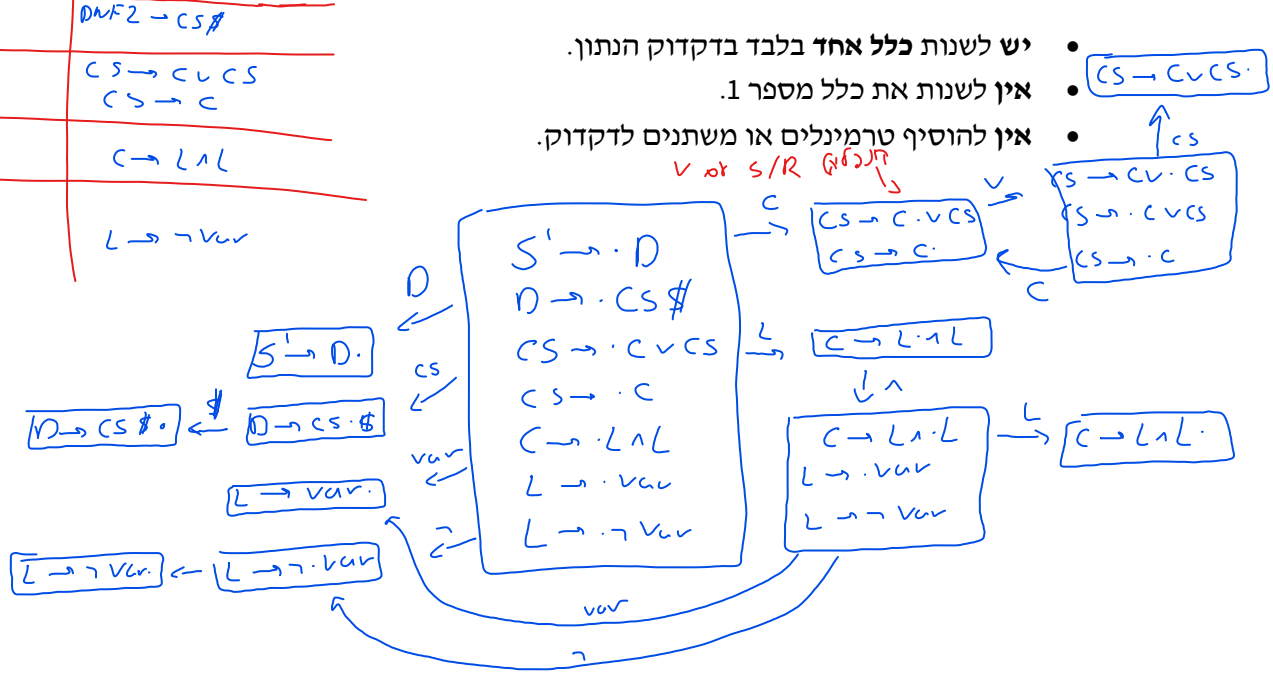
select(C → L ∧ L) = {var, ¬}
select(L → var) = {var}
select(L → ¬ var) = {¬}

א. (5 נק׳) האם הדקדוק שייך ל-LL(1)? הסבירו. לא קיים מבלבל בשורה CS
ב. (5 נק׳) האם הדקדוק שייך ל-LR(0)? הסבירו. קיים מבלבל S/R בקר אז לא
ג. (10 נק׳) שנו את הדקדוק על פי ההנחיות על מנת שתשובתכם לסעיף ב׳ תשתנה. הסבירו.

	var	¬
DNF2	DNF2 → CS$	DNF2 → CS$
CS	CS → C v CS CS → C	CS → C v CS CS → C
C	C → L ∧ L	C → L ∧ L
L	L → var	L → ¬ var

הנחיות:

- **יש** לשנות **כלל אחד** בלבד בדקדוק הנתון.
- **אין** לשנות את כלל מספר 1.
- **אין** להוסיף טרמינלים או משתנים לדקדוק.



שאלה 4: Backpatching (30 נק׳)

אתם עובדים בחברת המסתדרים, בשפת FanC, ולאחרונה הצטרף עובד חדש. העובד החדש התקשה ללמוד את FanC למרות שלמד במכללה טכנולוגית מאזור הצפון (לא חשדתם בו). העובד הסביר שהם למדו רק שפות עם ניהול שגיאות בצורה של מבנה בקרה try/catch ככה שבמידה ויש שגיאה בקוד שקורה תחת בלוק try ירוץ בלוק catch הכי קרוב. לדוגמא:

```
1   try {
2      try {
3         bar();
4         if (check_stuff()) {
5            error;
6         }
7         kochva();
8      } catch {
9         haya();
10     }
11     gibor();
12     if (check_other()) {
13        error;
14     }
15     hu();
16  } catch {
17     kara();
18  }
19  lidror();
```

ההרצה תתחיל בקריאה לפונקציה bar בשורה 3. במידה ותתרחש השגיאה בשורה 5, יבוצע הקוד בשורה 9 (haya) ואז הקוד בשורה 11 (gibor). אחרת יבוצע הקוד בשורה 7 (kochva) ולאחריו הקוד בשורה 11 (gibor).

באופן דומה אם תתרחש השגיאה בשורה 13 נריץ את הקוד בשורה 17 (kara) ואחריו את שורה 19 (lidror), אחרת נריץ את שורה 15 (hu) ולאחריה את שורה 19 (lidror).

אותו עובד ביקש שתוסיפו ל-FanC מאפיין (פיצ׳ר) כנ״ל (בקשה קצת חצופה, אבל לא חשדתם בו), לכן החלטתם להוסיף תמיכה למאפיין הזה באופן הבא:

- זריקת שגיאה תעשה מפורשות ע״י הוספת מילה שמורה לשפה, ״error״, שהתחביר שלה זהה לזה של return בלי ערך חזרה.
- כל שגיאה **תמיד** תופיע בתוך בלוק try כך שקיים בלוק catch בצמוד ל-try שתופס את השגיאה. זאת אומרת שכל פקודת error נמצאת ישירות בתוך בלוק try (אופציונלית תחת תנאי if או לולאה); בפרט, ה-error וה-catch נמצאים **באותה פונקציה**, ולכן אפשר לדעת בזמן הקומפילציה איזה בלוק catch רלוונטי לשגיאה.

נתונים כללי הדקדוק עבור מבנה הבקרה `try/catch`:

S → try LBRACE S RBRACE catch LBRACE S RBRACE

S → error SC

עבור סעיפים א׳ וב׳ ניתן להניח שאין שגיאות קומפילציה ושקיים ניהול זיכרון אוטומטי ולכן אין צורך לטפל בשחרור זיכרון.

א. (8 נק׳) הציעו פריסת קוד המתאימה לשיטת backpatching עבור מבנה הבקרה הנ״ל. על הקוד הנוצר להיות יעיל ככל האפשר. הסבירו מהן התכונות הסמנטיות שאתם משתמשים בהן עבור כל משתנה.

ב. (12 נק׳) כתבו סכימת תרגום בשיטת backpatching המייצרת את פריסת הקוד שהצעתם בסעיף הקודם. על הסכימה להיות יעילה ככל האפשר, הן מבחינת זמן הריצה שלה והן מבחינת המקום בזיכרון שנדרש עבור התכונות הסמנטיות.

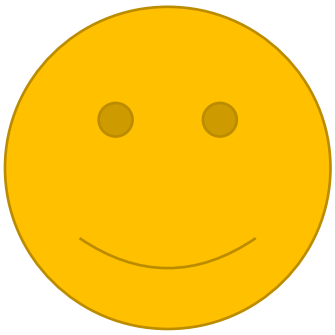
שימו לב:

- אין לשנות את הדקדוק, למעט הוספת מרקרים M, N שנלמדו בכיתה בלבד.
- אין להשתמש בכללים סמנטיים באמצע כלל גזירה.
- אין להשתמש במשתנים גלובליים בזמן קומפילציה.
- למשתנה S יש כללי גזירה פרט לאלו המוצגים בשאלה.

אחרי שסיימתם לממש את המאפיין הגיע אליכם מנכל חברת המסתדרים בכעס רב, הבן שלו (פה חשדתם) ביקש שהטיפול בשגיאות יהיה **דינאמי** ולא סטטי. הכוונה היא שבזמן ריצה יימצא בלוק catch הקרוב ביותר, כמו בשפת C++, והקוד שלו ירוץ. עתה פקודת error יכולה להופיע בכל מקום בקוד ולא ניתן לדעת בזמן קומפילציה איזה בלוק catch יתפוס את השגיאה.

ג. (10 נק׳) הסבירו בקצרה (ללא מימוש ועד 5 שורות) אילו שינויים יש לבצע בקומפיילר על מנת לתמוך במאפיין החדש. בתשובתכם התייחסו לניהול הזיכרון במחסנית.

סוף המבחן



נוסחאות ואלגוריתמים

כל ההגדרות מתייחסות לדקדוק $G = (V, T, P, S)$.

Top Down

$first(\alpha) = \{ t \in T \mid \alpha \Rightarrow^* t\beta \wedge \beta \in (V \cup T)^* \}$

$follow(A) = \{ t \in T \cup \{\$\} \mid S\$ \Rightarrow^* \alpha A t \beta \wedge \alpha \in (V \cup T)^* \wedge \beta \in (V \cup T)^* (\varepsilon \mid \$) \}$

$$select(A \to \alpha) = \begin{cases} first(\alpha) \cup follow(A) & \alpha \Rightarrow^* \varepsilon \\ first(\alpha) & \text{otherwise} \end{cases}$$

הגדרה: דקדוק G הוא LL(1) אם ורק אם לכל שני כללים ב- G השייכים לאותו משתנה A מתקיים:
$select(A \to \alpha) \cap select(A \to \beta) = \varnothing$

הגדרת טבלת המעברים $M : V \times (T \cup \{\$\}) \to P \cup \{error\}$ עבור דקדוק LL(1):

$$M[A , t] = \begin{cases} A \to \alpha & t \in select(A \to \alpha) \\ error & t \notin select(A \to \alpha) \text{ for all } A \to \alpha \in P \end{cases}$$

אלגוריתם מנתח LL(1):

```
Q.push(S)
while !Q.empty() do
     X = Q.pop()
     t = next token
     if X ∈ T then
          if X = t then MATCH
          else ERROR
     else // X ∈ V
          if M[X , t] = error then ERROR
          else PREDICT(X , t)
     end if
end while
t = next token
if t = $ then ACCEPT
else ERROR
```

Bottom Up

פריט LR(0) הוא $(A \to \alpha \bullet \beta)$ כאשר $A \to \alpha\beta \in P$

סגור (closure) על קבוצת פריטים I מוגדר באופן אינדוקטיבי:

- בסיס: $closure(I) = I$.
- צעד: אם $(A \to \alpha \bullet B\beta) \in closure(I)$, אז לכל $B \to \gamma \in P$, גם $(B \to \bullet\gamma) \in closure(I)$

פונקציית המעברים של האוטומט:

$$\delta(I , X) = \bigcup \left\{ closure(A \to \alpha X \bullet \beta) \mid (A \to \alpha \bullet X\beta) \in I \right\}$$

פריט LR(1) הוא $(A \to \alpha \bullet \beta , t)$ כאשר $A \to \alpha\beta \in P$, $t \in T \cup \{\$\}$

סגור (closure) על קבוצת פריטים I מוגדר באופן אינדוקטיבי:

- בסיס: $closure(I) = I$.
- צעד: אם $(A \to \alpha \bullet B\beta , t) \in closure(I)$, אז לכל $B \to \gamma \in P$ ולכל $x \in first(\beta t)$, גם $(B \to \bullet\gamma , x) \in closure(I)$

פונקציית המעברים של האוטומט:

$$\delta(I , X) = \bigcup \left\{ closure(A \to \alpha X \bullet \beta , t) \mid (A \to \alpha \bullet X\beta , t) \in I \right\}$$

הגדרת טבלת action למנתח SLR:

$$action[i , t] = \begin{cases} SHIFT_j & \delta(I_i ,t) = I_j \\ REDUCE_k & \text{rule k is } A \to \alpha,\ (A \to \alpha \bullet) \in I_i \text{ and } t \in follow(A) \\ ACCEPT & (S' \to S \bullet) \in I_i \text{ and } t = \$ \\ ERROR & \text{otherwise} \end{cases}$$

הגדרת טבלת action למנתח LR(1):

$$action[i , t] = \begin{cases} SHIFT_j & \delta(I_i ,t) = I_j \\ REDUCE_k & \text{rule k is } A \to \alpha \text{ and } (A \to \alpha \bullet , t) \in I_i \\ ACCEPT & (S' \to S \bullet , \$) \in I_i \text{ and } t = \$ \\ ERROR & \text{otherwise} \end{cases}$$

הגדרת טבלת goto למנתח SLR ו- LR(1):

$$goto[i , X] = \begin{cases} j & \delta(I_i , X) = I_j \\ error & \text{otherwise} \end{cases}$$

אלגוריתם מנתח shift/reduce:

```
Q.push(0)        // where 0 is the initial state of the prefix automaton
while true do
     k = Q.top().state
     t = next token
     do action[k , t]
end while
```

קוד ביניים

סוגי פקודות בשפת הביניים:

1. משפטי השמה עם פעולה בינארית
2. משפטי השמה עם פעולה אונרית
3. משפטי העתקה
4. קפיצה בלתי מותנה
5. קפיצה מותנה
6. הדפסה

```
x := y op z
x := op y
x := y
goto L
if x relop y goto L
print x
```

Data-Flow Analysis

ההגדרות מתייחסות ל – CFG: G=(V,E).

הצורה הכללית של המשוואות בחישוב סריקה קדמית:

$$in(B) = \bigcap_{(S,B)\in E} out(S) \quad \text{או} \quad in(B) = \bigcup_{(S,B)\in E} out(S)$$

$$out(B) = f_B\big(in(B)\big)$$

הצורה הכללית של המשוואות בחישוב סריקה אחורית:

$$out(B) = \bigcap_{(B,S)\in E} in(S) \quad \text{או} \quad out(B) = \bigcup_{(B,S)\in E} in(S)$$

$$in(B) = f_B\big(out(B)\big)$$

שפת FanC

אסימונים:

אסימון	תבנית
VOID	void
INT	int
BYTE	byte
B	b
BOOL	bool
SET	set
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	+ | - | * | /
DOTS	..
ID	[a-zA-Z][a-zA-Z0-9]*
NUM	0 | [1-9][0-9]*
STRING	"([^\n\r\"\\]|\\[rnt"\\])+"

דקדוק:

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. $FormalDecl \rightarrow EnumType\ ID$
13. $Statements \rightarrow Statement$
14. $Statements \rightarrow Statements\ Statement$
15. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
16. $Statement \rightarrow Type\ ID\ SC$
17. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
18. $Statement \rightarrow EnumType\ ID\ ASSIGN\ Exp\ SC$
19. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
20. $Statement \rightarrow Call\ SC$
21. $Statement \rightarrow RETURN\ SC$
22. $Statement \rightarrow RETURN\ Exp\ SC$
23. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
24. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
25. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
26. $Statement \rightarrow BREAK\ SC$
27. $Statement \rightarrow CONTINUE\ SC$
28. $Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
29. $Call \rightarrow ID\ LPAREN\ RPAREN$
30. $ExpList \rightarrow Exp$
31. $ExpList \rightarrow Exp\ COMMA\ ExpList$
32. $Type \rightarrow INT$
33. $Type \rightarrow BYTE$
34. $Type \rightarrow BOOL$
35. $Type \rightarrow SET\ LBRACKET\ NUM\ DOTS\ NUM\ RBRACKET$
36. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
37. $Exp \rightarrow Exp\ BINOP\ Exp$
38. $Exp \rightarrow ID$
39. $Exp \rightarrow Call$

40. $Exp \rightarrow NUM$
41. $Exp \rightarrow NUM\ B$
42. $Exp \rightarrow STRING$
43. $Exp \rightarrow TRUE$
44. $Exp \rightarrow FALSE$
45. $Exp \rightarrow NOT\ Exp$
46. $Exp \rightarrow Exp\ AND\ Exp$
47. $Exp \rightarrow Exp\ OR\ Exp$
48. $Exp \rightarrow Exp\ RELOP\ Exp$
49. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$