26.02.2019

מבחן סוף סמסטר – מועד ב'

מרצה אחראי: ד"ר שחר יצחקי

מתרגלים: יעקב סוקוליק, אלכסנדר סיבק, עומר כץ

הוראות:

א. בטופס המבחן 16 עמודים מהם 6 דפי נוסחאות. בדקו שכל העמודים ברשותכם.
ב. משך המבחן שלוש שעות (180 דקות).
ג. אסור כל חומר עזר חיצוני.
ד. במבחן 5 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 (20 נק'): שלבי הקומפילציה

שני חלקי השאלה מתייחסים לשפת FanC שהופיעה בתרגילי הבית.

חלק א - סיווג מאורעות (10 נקודות)

נתון קטע הקוד הבא בשפת FanC:

```
int sqr(int a) {
    return a * a;
}

int abs(int a) {
    if (a > 0)
        return a;
    else
        return 0 - a;
}

int foo(bool l2Norm, int x, int y) {
    if (l2Norm)
        return sqr(x) + sqr(y);
    else {
        return abs(x) + abs(y);
    }
}

void main() {
    struct Point {
        int x;
        int y;
    };
    struct Point p1;
    p1.x = 2;
    p1.y = 2;
    int res = foo(true, p1.x, p1.y);
    return;
}
```

בסעיפים הבאים מוצגים שינויים (בלתי תלויים) לקוד של התוכנית. עבור כל שינוי כתבו האם הוא גורם לשגיאה. אם כן, ציינו את השלב המוקדם ביותר שבה נגלה אותה (ניתוח לקסיקלי, ניתוח תחבירי, ניתוח סמנטי, ייצור קוד ,זמן ריצה) ונמקו בקצרה:

א. מחליפים את שורה 9 בשורה הבאה:

```
9.        return -a;
```

ב. מחליפים את שורה 12 בשורה הבאה:

```
12.  int foo(bool l2Norm, int x, int y, struct Point p) {
```

ג. מחליפים את שורה 28 בשורה הבאה:

```
28.  int res = foo(true, p1.x, p1.y, p1);
```

ד. מחליפים את שורה 26 בשורה הבאה:

```
26.    p1.x = p1.y = 2;
```

ה. מוחקים את שורה 26.

חלק ב – הרחבת השפה (10 נקודות)

הנכם מתבקשים להוסיף **לשפת FanC** יכולת חדשה. קראו את תיאור היכולת, ופרטו בקצרה איזה שינוי צריך להתבצע בכל שלב בקומפילציית השפה. **התייחסו לשלבים לקסיקלי, תחבירי, סמנטי, ייצור קוד אסמבלי (שפת ביניים)**. הקפידו על ההפרדה בין השלבים. יש להקפיד על פתרון **יעיל**.

נרצה להוסיף לשפת FanC את היכולת להגדיר תבניות (templates) של פונקציות, בדומה למנגנון ב-++C.
תבניות מאפשרות להגדיר פונקציה המשתמשת בטיפוס גנרי יחיד, אשר יסומן ע"י המילה השמורה T.
לאחר הגדרה של תבנית, ניתן להשתמש בה בכל מקום בתוכנית תוך ציון הטיפוס המפורש אשר מחליף את T. להלן התחביר ודוגמא להמחשה:

```
template <typename T>
T genericAdd(T a, T b) {
        return a + b;
}

int main() {
        int resA = genericAdd<int>(255, 1); // 256
        byte resB = genericAdd<byte>(255 b, 1 b)); // overflows, thus 0
        printi(resA); printi(resB);
}
```

השימוש בפונקציה גנרית אפשרי אך ורק אם היא כבר הוגדרה קודם לכן בקוד (כמו פונקציה רגילה).

אם בתבנית קיימת שגיאה סינטקטית, יש לדווח על כך מיד ולעצור את תהליך הקומפילציה.
אם בתבנית קיימת שגיאה סמנטית (או נוצרת אחת כזאת בעקבות שימוש בטיפוס קונרטי מסויים בתבנית), יש לדווח על כך רק לאחר instantialization של התבנית (עם אותו טיפוס קונקרטי).
ציינו מתי הקומפיילר ידווח על השגיאה ויעצור את הקומפילציה בהתאם למימוש שלכם.

שימו לב – בניגוד לדרישות בשיעורי הבית, אין דרישה שהניתוח התחבירי והסמנטי יתבצעו במעבר יחיד על עץ הגזירה.

שאלה 2 (30 נקודות): DFA

בשפה KennY יש רק שני סוגים של משפטי השמה: $x := n$ ו- $x := y + n$, כאשר n מספר שלם ו-x, y הם משתנים (יכולים להיות אותו משתנה).

בנוסף ישנם תנאים בוליאניים ומשפטי בקרה בדומה לשפת WHILE. תחביר מלא של השפה נתון על-ידי הדקדוק הבא:

$$
\begin{aligned}
S &\rightarrow x := n \mid x := y + n \mid \text{skip} \\
&\mid \text{if } E \text{ then } S \text{ else } S \\
&\mid \text{while } E \text{ do } S \\
E &\rightarrow x \diamond y \qquad (\diamond \in \{<, >, \leq, \geq, =, \neq\})
\end{aligned}
$$

קני (ממציא השפה) רוצה לעקוב אחרי כל הערכים האפשריים של כל משתנה בתכנית, ולכן בחר להשתמש בסריג קבוצת החזקה של המספרים השלמים $L = \langle \mathcal{P}(\mathbb{Z}), \subseteq \rangle$. לשם כך הוא כתב את פונקציות המעבר הבאות:

s	$[\![s]\!]^{\#}\sigma^{\#}$
$x := n$	$\sigma^{\#}[x \mapsto \{n\}]$
$x := y + n$	$\sigma^{\#}[x \mapsto \{v + n \mid v \in \sigma^{\#}(y)\}]$
skip	$\sigma^{\#}$
if e	$\sigma^{\#}$

(הערה: זכרו שמצב אבסטרקטי $\sigma^{\#}$ שייך לסריג חזקה L^k – עבור תכנית עם k משתנים – ויש בו ערך אבסטרקטי אחד לכל משתנה בתכנית; וכן שהפעולה $\sigma^{\#}[x \mapsto e]$ מחליפה את הערך שמתאים למשתנה x במצב $\sigma^{\#}$ לערך אבסטרקטי חדש e.)

א. הראו שפונקציות המעברים של קני הן מונוטוניות.

ב. הראו תכנית דוגמה שעבורה הרצת האנליזה של קני לא עוצרת.

ג. כדי להתגבר על בעיית אי-העצירה, קני מוכן להתפשר ולעקוב אחרי **שלושה ערכים שונים לכל היותר** עבור כל משתנה בתכנית. הערכים אינם ידועים מראש ויכולים להיות שונים מתכנית לתכנית וכן בין משתנים שונים באותה תכנית.

משמעות הדבר היא, שאם בנקודה כלשהי בתכנית משתנה מסוים יכול לקבל (על פי האנליזה המקורית) ארבעה ערכים שונים או יותר, אז באותה נקודה ניתן יהיה להניח (באנליזה החדשה) שאותו משתנה יכול לקבל *כל ערך שלם שהוא*.

(1) הגדירו את הסריג המתאים, את יחס הסדר ($\sqsubseteq$) ואת אופרטור ה-join ($\sqcup$).

(2) הגדירו פונקציות מעבר חדשות והראו שהן מונוטוניות.

(3) הסבירו מדוע האנליזה המתוקנת **תמיד עוצרת**.

(הערה חשובה: קני מתעלם מערכי האמת של תנאים בולאניים באנליזה שלו, ולכן גם באנליזה המתוקנת מותר לעשות כך.)

שאלה 3 (10 נקודות): אופטימיזציות

JavaScript הינה שפת תכנות עילית ו-dynamically typed המשמשת בין השאר להצגת דפי אינטרנט אינטראקטיביים. דפדפני אינטרנט מודרניים מכילים מנוע JIT מתקדם המריץ את קוד המקור הנשלח ע"י השרת אצל המשתמש הנכנס לאתר.

לאחרונה, הוצע סטנדרט חדש לדפדפנים בשם WebAssembly. נתאר אותו כאן בתמציות (ועם מספר אי דיוקים). הסטנדרט מגדיר פורמט בינארי לייצוג של חסכוני של קוד ביניים הקרוב מאוד לקוד מכונה. הסטנדרט מאפשר לקמפל קוד משפה אחרת, למשל C++, לייצוג של קוד ביניים ולהעבירו לדפדפן – שם הוא יתקמפל מיד לקוד מכונה ישירות (תוך עקיפת מנגנון ה-JIT).

1. (6 נק') הציגו שני יתרונות אפשריים עבור שימוש של WebAssembly ביחס ל-JavaScript, על בסיס המידע הנתון. פרטו ונמקו.

2. (2 נק') מדוע המידע המועבר הינו ייצוג של קוד ביניים, ולא קוד מכונה סופי מוכן לריצה?

3. (2 נק') תארו אופטימיזציה אחת שהדפדפן יוכל לבצע עבור קוד JavaScript באמצעות מנוע ה-JIT, אך לא תתאפשר באמצעות WebAssembly בתצורה הנתונה.

שאלה 4 (15 נק'): ניתוח תחבירי וסמנטי

נתון דקדוק Reverse Polish Notation עבור אריתמטיקה. ב-Reverse Polish Notation, מופיעים קודם האופרנדים, ורק אחריהם האופרטור לפי סדר ההפעלה שלהם. (אסימונים מופיעים בדקדוק עם קו תחתון, משתנים ללא קו תחתון):

1. $E \rightarrow E\ E\ Op$
2. $E \rightarrow \underline{num}$
3. $Op \rightarrow \underline{+}$
4. $Op \rightarrow \underline{-}$
5. $Op \rightarrow \underline{/}$

הניחו כי לאסימון num קיימת התכונה הסמנטית val המכילה את ערך המספר.

א. (4 נק') האם הדקדוק שייך ל-LR(0)? האם הוא שייך ל-SLR?

ב. (3 נק') הוכיחו כי הדקדוק אינו שייך ל-LL(1).

ג. (3 נק') תקנו את הדקדוק כך שהוא יהיה שייך ל-LL(1) והוכיחו כי הוא אכן שייך ל-LL(1).

ד. (5 נק') נרצה לבצע על ביטויים בשפה (הנתונה) את הבדיקה הסמנטית הבאה: במקרה של חלוקה בקבוע 0 נרצה להחזיר שגיאה.
כתבו כללים סמנטיים לדקדוק הנתון (ללא התיקון שביצעתם בסעיף ג) לצורך בדיקת התכונה הסמנטית לעיל בזמן ניתוח.
במידה והתבצעה חלוקה ב-0 יש לזרוק חריגה כך:

$throw\ new\ Error(\text{“}div\ by\ zero\text{”});$

השתמשו **בתכונות נוצרות בלבד**. הסבירו מתי במהלך ריצת המנתח הכללים הסמנטיים יופעלו.
הנחיות:

- אין לשנות את הדקדוק.
- יש לבצע את הניתוח הסמנטי בזמן בניית עץ הגזירה.
- ניתן להוסיף למשתנים תכונות סמנטיות כרצונכם. יש לציין אותן מפורשות.
- אין להשתמש במשתנים גלובליים.
- יש לכתוב את הכללים הסמנטיים במלואם.

שאלה 5 (25 נקודות): Code Generation

חלק א (10 נקודות) – טיפול בשגיאות

למדנו בכיתה שאחת הדרכים לממש טיפול בחריגות (exceptions) היא באמצעות שמירת כתובת הexception handler הנוכחי ברשומת ההפעלה. בפתרון זה, בעת זריקת חריגה, נבדוק אם ה exception handler הנוכחי מטפל בחריגה שנזרקה. אם כן, נבצע אותו. אם לא, נמצא את רשומת ההפעלה הקודמת לרשומת ההפעלה הנוכחית, נמצא את הexception handler ששמור בה ונבדוק האם הוא מטפל בחריגה. נמשיך לחפש כך עד שימצא exception handler מתאים (או שנגיע לdefault exception handler).

טימי לא מרוצה מכך שבמנגנון טיפול בחריגות זה אנו נאלצים לבזבז במשאבים ב"חיפוש" ה exception handler הנכון בין רשומות ההפעלה שבמחסנית. טימי מציע לשנות את אופן קימפול exception handlers כך שכל exception handler יכיל בתוכו את כל הexception handler שקודמים לו. לטענת טימי, בפתרון הזה תמיד יספיק לבדוק רק את הexception handler הנוכחי.

לצורך מימוש פתרון זה, נניח שקיימת מחלקת חריגה Exception ממנה יורשות כל מחלקות החריגה האחרות.

כלומר, בהנתן הקוד הבא:

```
1.  void f (int x) {
2.     try {
3.        int y = 0;
4.        try {
5.           if ( x > y ) {
6.              <throw exception>
7.           }
8.        }
9.        catch (Exception1 e) {
10.          <handle exception e1>
11.       }
12.    }
13.    catch (Exception2 e) {
14.       <handle exception e2>
15.    }
16. }
17.
18. void main() {
19.    f();
20. }
```

הקומפיילר ייצר קוד ששקול ל:

```
1'.  void f (int x) {
2'.     try {
3'.        int y = 0;
4'.        try {
5'.           if ( x > y ) {
6'.              <throw exception>
7'.           }
8'.        }
9'.        catch (Exception e) {
10'.          if (typeof(e) is Exception1) {
11'.             <exception handler of line 10>
12'.          }
13'.          if (typeof(e) is Exception2) {
14'.             <exception handler of line 14>
15'.          }
16'.          abort(); // crashes program
17'.       }
18'.    }
19'.    catch (Exception e) {
20'.          if (typeof(e) is Exception2) {
21'.             <exception handler of line 14>
22'.          }
23'.          abort(); // crashes program
24'.    }
25'. }
26'.
27'. void main() {
28'.    f();
29'. }
```

(Exception1 ו Exception2 יורשות מException)

א. (6 נק') הפתרון המוצע אינו יכול לעבוד בתצורה זו. הסבירו את הבעיה בפתרון של טימי והציעו תכנית שמדגימה את הבעיה.

ב. (4 נק') ברצוננו "להרוויח" כמה שיותר מההצעה של טימי. הציעו תיקון פשוט ככל האפשר להצעה של טימי שיפתור את הבעיה הקיימת בהצעה הנוכחית.
שימו לב: יתכן והפתרון שתציעו לא יחסוך לגמרי את החיפוש בין רשומות ההפעלה (כפי שצויין בתחילת השאלה), אך עליו לחסוך כמה שיותר.

חלק ב (15 נקודות) – חלוקת רגיסטרים לCallee-saved וCaller-saved

נתון הקטע קוד הבא:

```
int f(int x, int y) {
        int a,b,c;
        a = x + 2;
        b = a * y;
        c = y – 3;
        return a * b * c;
}
int g() {
        int x,y,z,w;
        x = 13;
        y = 42;
        z = f(x,y);
        w = f(y,x);
        y = f(w,x);
        x = f(y,z);
        return x + y;
}
```

הקומפיילר החליט להקצות רגיסטרים למשתנים באופן הבא:

- x -> r0
- y -> r1
- z -> r2
- w -> r3
- a -> r0
- b -> r1
- c -> r4
- f.x -> r5
- f.y -> r3

א. (9 נק') הציעו חלוקה של הרגיסטרים לcaller-saved וcallee-saved שתצמצם למינימום את מספר הכתיבות והקריאות של משתנים למחסנית בסך כל הקריאות למתודה f מתוך g.

ב. (6 נק') באפשרותכם להחליף את הרגיסטר המוקצה לאחד המשתנים עם רגיסטר אחר מבין הרגיסטרים הקיימים. הציעו החלפה שתצמצם עד כמה שניתן את מספר הכתיבות והקריאות של משתנים למחסנית בקריאות למתודה f.

בהצלחה!

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

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:

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

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:

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

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	**x := y op z**
2. משפטי השמה עם פעולה אונרית	**x := op y**
3. משפטי העתקה	**x := y**
4. קפיצה בלתי מותנה	**goto L**
5. קפיצה מותנה	**if x relop y goto L**
6. הדפסה	**print x**

Data-Flow Analysis

ההגדרות מתייחסות ל – CFG: G=(V,E).

הצורה הכללית של המשוואות בחישוב סריקה קדמית :

$$in(B) = \bigcap out(S) \text{ או } in(B) = \bigcup out(S)$$
$$out(B) = f_B(in(B))$$

הצורה הכללית של המשוואות בחישוב סריקה אחורית :

$$out(B) = \bigcap in(S) \text{ או } out(B) = \bigcup in(S)$$
$$in(B) = f_B(out(B))$$

שפת FanC

אסימונים:

אסימון	תבנית
VOID	void
INT	int
BYTE	byte
B	b
BOOL	bool
STRUCT	struct
AND	and
OR	or
NOT	not
TRUE	true
FALSE	false
RETURN	return
IF	if
ELSE	else
WHILE	while
BREAK	break
CONTINUE	continue
SC	;
COMMA	,
PERIOD	.
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״\\])+״

דקדוק:

1. $Program \rightarrow Structs\ Funcs$
2. $Funcs \rightarrow \epsilon$
3. $Funcs \rightarrow FuncDecl\ Funcs$
4. $FuncDecl \rightarrow RetType\ ID\ LPAREN\ Formals\ RPAREN\ LBRACE\ Statements\ RBRACE$
5. $Structs \rightarrow \epsilon$
6. $Structs \rightarrow StructsDecl\ Structs$
7. $StructsDecl \rightarrow STRUCT\ ID\ LBRACE\ StructMemList\ RBRACE\ SC$
8. $RetType \rightarrow Type$
9. $RetType \rightarrow VOID$
10. $Formals \rightarrow \epsilon$
11. $Formals \rightarrow FormalsList$
12. $FormalsList \rightarrow FormalDecl$
13. $FormalsList \rightarrow FormalDecl\ COMMA\ FormalsList$
14. $FormalDecl \rightarrow Type\ ID$
15. $FormalDecl \rightarrow StructType\ ID$
16. $StructMemList \rightarrow StructMem$
17. $StructMemList \rightarrow StructMem\ StructMemList$
18. $StructMem \rightarrow Type\ ID\ SC$
19. $Statements \rightarrow Statement$
20. $Statements \rightarrow Statements\ Statement$
21. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
22. $Statement \rightarrow Type\ ID\ SC$
23. $Statement \rightarrow StructType\ ID\ SC$
24. $Statement \rightarrow STRUCT\ ID\ LBRACE\ StructMemList\ RBRACE\ SC$
25. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
26. $Statement \rightarrow StructType\ ID\ ASSIGN\ Exp\ SC$
27. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
28. $Statement \rightarrow ID\ PERIOD\ ID\ ASSIGN\ Exp\ SC$
29. $Statement \rightarrow Call\ SC$
30. $Statement \rightarrow RETURN\ SC$
31. $Statement \rightarrow RETURN\ Exp\ SC$
32. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
33. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
34. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
35. $Statement \rightarrow BREAK\ SC$
36. $Statement \rightarrow CONTINUE\ SC$
37. $Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
38. $Call \rightarrow ID\ LPAREN\ RPAREN$

39. $ExpList \rightarrow Exp$
40. $ExpList \rightarrow Exp\ COMMA\ ExpList$
41. $Type \rightarrow INT$
42. $Type \rightarrow BYTE$
43. $Type \rightarrow BOOL$
44. $StructType \rightarrow STRUCT\ ID$
45. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
46. $Exp \rightarrow Exp\ BINOP\ Exp$
47. $Exp \rightarrow ID$
48. $Exp \rightarrow ID\ PERIOD\ ID$
49. $Exp \rightarrow Call$
50. $Exp \rightarrow NUM$
51. $Exp \rightarrow NUM\ B$
52. $Exp \rightarrow STRING$
53. $Exp \rightarrow TRUE$
54. $Exp \rightarrow FALSE$
55. $Exp \rightarrow NOT\ Exp$
56. $Exp \rightarrow Exp\ AND\ Exp$
57. $Exp \rightarrow Exp\ OR\ Exp$
58. $Exp \rightarrow Exp\ RELOP\ Exp$