29.1.2018

מבחן סוף סמסטר – מועד א'

מרצה אחראי: ד"ר שחר יצחקי

מתרגלים: אבנר אליזרוב, עומר כץ, הילה פלג

הוראות:

א. בטופס המבחן 13 עמודים מהם 5 דפי נוסחאות. בדקו שכל העמודים ברשותכם.

ב. משך המבחן שלוש שעות (180 דקות).

ג. במבחן 6 שאלות. כל השאלות הינן חובה. משקל כל שאלה מופיע בראשיתה. (חלוקת המשקל בין הסעיפים בכל שאלה אינה בהכרח אחידה.)

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

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

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

ז. אין צורך להגיש את הטופס בתום הבחינה.

ח. 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:

```
1.  int func1(int a, int c){
2.      return a / c;
3.  }
4.  int func2(){
5.      int a = 8;
6.      int c = 4;
7.      return func1(a, a - c) + a;
8.  }
9.  void main(){
10.     //comment
11.     int res = func2();
12.     return;
13. }
```

בסעיפים הבאים מוצגים שינויים (בלתי תלויים) ל-main 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**:

1. מחליפים את שורה 10 בשורה הבאה:

```
10. /*comment*/
```

2. מחליפים את שורה 6 בפקודה הבאה:

```
6. int c = a * 1;
```

3. מחליפים את שורה 10 בפקודה הבאה:

```
10. int[] a;
```

4. מחליפים את שורה 5 בפקודה הבאה:

```
5. byte a = 8;
```

5. מחליפים את שורה 1 בשורה הבאה:

```
1. int func1(int a){
```

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**.

שפת FanC מאפשרת לנו לתת אותו טיפול למספר casеים בפקודת switch באופן הבא:

```
switch (x) {
     case 1:
     case 2:
     case 3: <some code>
     …
};
```

נרצה לאפשר תמיכה קלה יותר כך שאת הקוד מעלה נוכל לשכתב כ:

```
switch (x) {
     case 1~3: <some code>
     …
};
```

הערכים החוקיים לcase הם ערך מספרי בודד או טווח ערכים מספריים מכל טיפוס נומרי. מותר, כמו במקרה של ערך בודד, לשרשר מספר case-ים ורק אחריהם פקודות.

טווח ערכים נכתב כmin~max. ניתן להשתמש בכל ליטרל נומרי, כך למשל כל הטווחים הבאים חוקיים:

```
switch(x) {
     case 1b~3: <some code>
     case 16~20b:
     case 100b~200b:
     <more code>
};
```

בהנתן טווח ערכים יש לוודא כי הטווח חוקי. טווח אינו חוקי אם אינו מכיל אף ערך. לדוגמה, 3~7 ו4~4 הם טווחים חוקיים (4~4 מכיל את האיבר הבודד 4) ואילו הטווח 8~6 אינו חוקי.

אם ערך הביטוי Exp במשפט ה-switch יהיה מספר המוכל בטווח, יבוצע כל הקוד שאחרי משפט ה-case עד ל-break או סוף ה-switch. כלומר, ההגדרה זהה לאוסף משפטי case עבור כל ערך בטווח.

שאלה 2 (30 נקודות): אנליזה סטטית

בשפת WHILE (ר' דקדוק להלן) כל המשתנים הם מטיפוס מספר שלם מתמטי (לא חסום). לפיכך, כדי לשמור על הסמנטיקה הנכונה, הקומפיילר נאלץ להקצות לכל ערך מספרי בודד כמות לא ידועה מראש של זכרון. פעולות במספרים גדולים כאלה הן מאד לא יעילות וישנה תקורת זכרון הנובעת מהצורך בהקצאה דינמית.

$$
\begin{aligned}
S &\rightarrow \text{id} := E \mid S\,;\,S \mid \text{skip} \\
&\mid \text{if } E \text{ then } S \text{ else } S \\
&\mid \text{while } E \text{ do } S \\
E &\rightarrow \text{id} \mid E \;\square\; E \qquad (\square \in \{+, -, *, /, =, \neq, <, >, \leq, \geq\})
\end{aligned}
$$

הערות לכלל הסעיפים:

- ניתן להשתמש בכל אנליזה סטטית שנלמדה בכיתה בציון שמה ודרך השימוש בה.
- ניתן להגדיר שורה בודדת כבלוק בסיסי (יש לציין זאת).
- יש לציין מהו ה-abstract domain, ומהן פונקציות האבסטרקציה והקונקרטיזציה. יש להגדיר את הסמנטיקה האבסטרקטית המתאימה. ניתן להפנות להגדרות מחומר הקורס ואין צורך להגדירן מחדש.

א. (10 נק') הציעו דרך המתבססת על אנליזה סטטית שבאמצעותה הקומפיילר יוכל להקצות את המשתנים המופיעים בתכנית (או את חלקם) באופן סטטי. הניחו שבשפת המטרה קיימים הטיפוסים הבסיסיים הבאים:

int8	$-2^7 .. 2^7-1$
uint8	$0 .. 2^8-1$
int16	$-2^{15} .. 2^{15}-1$
uint16	$0 .. 2^{16}-1$
int32	$-2^{31} .. 2^{31}-1$
uint32	$0 .. 2^{32}-1$

מצורפת תכנית לדוגמה (אין חובה להדגים את ריצת האנליזה על התכנית).

```
if (n <= 128) then (
     i := 1;
     sum := 0;
     while (i < n) do (
          prod := 2 * i * (i + 1) ;
          sum := sum + prod
          i := i + 1;
     )
)
else skip
```

ב. (5 נק') קיימים מצבים בהם בזמן ריצה ערך המשתנה לא יחרוג מגודל טיפוס מסוים אך לא ניתן לקבוע זאת בזמן קומפילציה בעזרת האנליזה שמימשתם בסעיף א'. הסבירו מדוע והדגימו בעזרת תכנית.

נרצה לאפשר אופטימיזציה דומה עבור מערכים. לשם כך מרחיבים את שפת WHILE שתכלול הקצאת מערכים (חד-ממדיים), גישות והשמות לאיברים.

S → id[E] := E

E → id[E] | new [E]

הסמנטיקה הקונקרטית של שפת WHILE המורחבת אינה מאפשרת השמה מחוץ לטווח האינדקסים המוגדר (0..length-1), וזאת באמצעות בדיקה בזמן ריצה. כמו כן, השמה של מערך למערך מעתיקה את כתובת המערך.

מצורפת תכנית לדוגמה (אין חובה להדגים את ריצת האנליזה על התכנית).

```
if (n <= 128) then (
    i := 1;
    a := new [n-1];
    while (i < n) do (
        a[i-1] := 2 * i * (i + 1);
        i := i + 1;
    )
    b := a;
    i := 1;
    sum := 0;
    while (i < n) do (
        sum := sum + b[i];
        b[i] := -b[i]
        i := i + 1;
    )
)
else skip
```

ג. (5 נק') הסבירו מדוע **לא ניתן** לבצע את האופטימיזציה באותה צורה כמו בסעיף א'.

ד. (10 נק') הציעו שיפור לאנליזה שתאפשר להגביל גם את הגודל של איברים במערך. שימו לב שהמערכים עצמם מוקצים דינמית (באמצעות new), אבל כאשר האיברים הם מטיפוס בסיסי ידוע מראש ניתן לבצע הקצאה יחידה עבור כל המערך ואין צורך להקצות זכרון בנפרד לכל איבר.

ניתן להניח שהתכנית תקינה מבחינת טיפוסים; כלומר, יש קבוצה של משתנים שמקבלים ערכים מספריים וקבוצה זרה ממנה של משתנים שמקבלים ערכים מטיפוס מערך, וכל ההשמות מכבדות את החלוקה הזאת.

שאלה 3 (10 נקודות): אופטימיזציה

נתון קוד הביניים הבא, שהוא גוף של פונקציה foo :

```
1.  c:=3
2.  d:=4
3.  c:=a+a
4.  b:=7
5.  d:=c*b
6.  a:=c
7.  if(?) goto 9
8.  c:=a
9.  d:=7
10. b:=d
11. if(?) goto 13
12. a:=5;
13. goto 15
14. goto 9
15. f:=c+a
16. e:=6
17. b:=a
18. if (?) goto 19
19. f:=e+b
20. d:=a
21. c:=2
22. a:=5
23. d:=c
24. f:=b+c
```

א. (3 נק') ציירו CFG לקוד הנתון כאשר כל צומת הוא בלוק בסיסי מקסימלי לפי האלגוריתם שנלמד בכיתה (ניתן להשתמש במספרי שורות במקום לכתוב את השורה עצמה).

ב. (3 נק') נתונות תוצאות אנליזת Liveness שנערכה על המתודה foo עבור הבלוק הבסיסי שמכיל את שורה 1 (B1) ועבור הבלוק הבסיסי שמכיל את שורה 24 (B24) :

$In_{liveness}(B1) = \{a\}$
$Out_{liveness}(B1) = \{a, b\}$
$In_{liveness}(B24) = \{\}$
$Out_{liveness}(B24) = \{a, b, e\}$

בהינתן המידע הנ"ל, בצעו אופטימיזציית Useless Code Elimination עבור הבלוקים B1 ו-B24. **כתבו** את הבלוקים המתקבלים לאחר הפעלת האופטימיזציה. **אין** לבצע אופטימיזציות נוספות.

ג. (4 נק') נשים לב כי בשורה 13 ובשורה 18 קיימות **קפיצות מיותרות** - קפיצות שאם נמחק אותן ונאחד את בלוק המקור ובלוק היעד שלהן לבלוק יחיד משמעות התכנית לא תשתנה. הציעו אנליזה גלובלית אשר בהינתן CFG תזהה **קפיצות מיותרות**.

שאלה 4 (15 נק'): ניתוח תחבירי וסמנטי

נתון הדקדוק הבא מעל האסימונים lpar (סוגריים שמאליים), rpar (סוגריים ימניים), id (שם משתנה) , num (מספר) ו-comma (פסיק). אסימונים מופיעים בדקדוק עם קו עליון, משתנים ללא קו עליון:

$$S \rightarrow \overline{\text{lpar}}\, L\, \overline{\text{rpar}}$$
$$L \rightarrow \overline{\text{id}}\, \overline{\text{comma}}\, L$$
$$L \rightarrow \overline{\text{num}}$$

לאסימון num תכונה סמנטית בשם value המכילה את הערך המספרי, ולאסימון id תכונה בשם name המכילה את שם המשתנה. אין לאסימונים תכונות סמנטיות נוספות.

נרצה לבדוק על מילה בשפה את התכונה הסמנטית:
מספר המשתנים המופיעים ברשימה **ללא חזרות** שווה למספר המופיע בסוף הרשימה.

כך למשל אלו רשימות תקינות סמנטית:

(0)
(a , 1)
(a , b , c , b , a , 3)

ורשימות אלו אינן תקינות סמנטית:

(a , 0)
(a , a , b , c , 4)

א. (2 נק') הראו כי הדקדוק הוא LR(0).
ב. (5 נק') כתבו כללים סמנטיים לבדיקת התכונה הסמנטית לעיל בזמן ניתוח LR(0). במידה והבדיקה נכשלה, יש להשתמש בפונקציה error() שתסיים את ריצת המנתח כדי לדווח על השגיאה מוקדם ככל הניתן.
השתמשו בתכונות **נוצרות בלבד** או תכונות **נורשות בלבד**. ציינו באילו השתמשתם, והסבירו 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.

ג. (2 נק') הראו כי הדקדוק הוא LL(1).
ד. (6 נק') פתרו את סעיף ב' עבור ניתוח LL(1).

שאלה 5 (10 נקודות): רשומות הפעלה

א. נתון קוד שמורכב מהרבה פרוצדורות קצרות. קיים חשש שכתיבת כתובת החזרה למחסנית בכל קריאה וקריאתה מהמחסנית בסיום הפרוצדורה יגרמו להאטה בריצת התכנית. מאחר וקריאה/כתיבה לרגיסטר מהירות מקריאה/כתיבה למחסנית, הוצע לשמור את כתובת החזרה ברגיסטר על מנת להאיץ את התכנית.

i. (3 נק') בהנתן שהפרוצדורות קוראות אחת לשניה, האם שינוי זה יאיץ את התכנית? הסבירו.

ii. (3 נק') בהנתן שקיימת פרוצדורה מרכזית אחת ואך ורק היא קוראת לפרוצדורות האחרות, האם השינוי יאיץ את התכנית? הסבירו.

ב. (4 נק') נתון מעבד בעל כמות בלתי מוגבלת של רגיסטרים. כדי לנצל יכולת זו, נרצה להקצות בשלב יצירת הקוד אוסף רגיסטרים נפרד לכל פרוצדורה, ובכך לחסוך את גיבוי ערכי הרגיסטרים ואת המקום שמוקצה להם ברשומת ההפעלה. האם שינוי זה ישמור על נכונות התכנית? הסבירו.

שאלה 6 (15 נקודות): Backpatching

נתון כלל הדקדוק עבור מבנה הבקרה for2:

$S \rightarrow \text{for2}\ (S_1; B; S_2)\ S_3;$

המבנה for2 יבצע את המשפט S_3 מספר פעמים לפי הסמנטיקה הבאה:
ראשית יתבצע משפט האתחול S_1, ואחריו יתבצעו בלולאה בדיקת התנאי B, המשפט S_3, ומשפט העדכון S_2 (בדומה ללולאת for רגילה). לולאת for2 תסתיים אחרי שיערוך התנאי B כ-false במהלך שתי איטרציות רצופות של הלולאה.

לדוגמא, עבור הקוד:

```
for2(int i = 0; i != 3 && i < 5; i++)
     print(i);
```

תודפס התוצאה:

```
0
1
2
3
4
5
```

א. (5 נק') הציעו פריסת קוד המתאימה לשיטת backpatching עבור מבנה הבקרה הנ"ל. על הקוד הנוצר להיות יעיל ככל האפשר. הסבירו מהן התכונות שאתם משתמשים בהן עבור כל משתנה.

ב. (10 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 N,M שנלמדו בכיתה בלבד.
- אין להשתמש בכללים סמנטיים באמצע כלל גזירה.
- אין להשתמש במשתנים גלובליים בזמן קומפילציה.
- למשתנים S,B ישנן התכונות שהוגדרו בכיתה בלבד.
- למשתנים S,B יש כללי גזירה פרט לאלו המוצגים בשאלה.

בהצלחה!

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

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

<u>פריט LR(0)</u> הוא $(A \to \alpha \bullet \beta)$ כאשר $A \to \alpha\beta \in P$
<u>סגור (closure)</u> 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 $(B \to \bullet\gamma) \in closure(I)$

<u>פונקציית המעברים של האוטומט:</u>

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

<u>פריט LR(1)</u> הוא $(A \to \alpha \bullet \beta , t)$ כאשר $A \to \alpha\beta \in P$, $t \in T \cup \{\$\}$
<u>סגור (closure)</u> 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 $(B \to \bullet\gamma , x) \in closure(I)$

<u>פונקציית המעברים של האוטומט:</u>

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

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
```

ייצור קוד בשיטת Backpatching

פונקציות:

`makelist(quad)`	יוצרת רשימה ריקה עם איבר אחד (ה"חור" quad).
`merge(list1,list2)`	מחזירה רשימה ממוזגת של הרשימות list1, list2
`emit(code string)`	מדפיסה קוד בשפת הביניים ומאפשרת להדפיס פקודות קפיצה עם "חורים".
`nextquad()`	מחזירה את כתובת הרביעיה (הפקודה) הבאה שתצא לפלט.
`backpatch(list, quad)`	מקבלת רשימת "חורים" list וכתובת quad, ו"מטליאה" את הרשימה כך שבכל החורים תופיע הכתובת quad.
`newtemp()`	מחזירה שם של משתנה זמני חדש שאינו נמצא בשימוש בתכנית.

משתנים סטנדרטיים:

- S: גוזר פקודות (statements) בשפה. תכונות:
  - nextlist: רשימת כתובות של פקודות המכילות חור שיש להטליא בכתובת הפקודה הבאה לביצוע אחרי הפקודה הנגזרת מ- S.
- B: גוזר ביטויים בוליאניים. תכונות:
  - truelist: רשימת כתובות של פקודות המכילות חור שיש להטליא בכתובת אליה יש לקפוץ אם הביטוי הבוליאני מתקיים.
  - falselist: רשימת כתובות של פקודות המכילות חור שיש להטליא בכתובת אליה יש לקפוץ אם הביטוי הבוליאני אינו מתקיים.
- E: גוזר ביטויים אריתמטיים. תכונות:
  - E.place: שם המשתנה הזמני לתוכו מחושב הביטוי האריתמטי.

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`

שפת FanC

אסימונים:

אסימון	תבנית
VOID	void
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
SWITCH	switch
CASE	case
DEFAULT	default
BREAK	break
COLON	:
SC	;
COMMA	,
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:

$Program \rightarrow Funcs$
$Funcs \rightarrow \epsilon$
$Funcs \rightarrow FuncDecl\ Funcs$
$FuncDecl \rightarrow RetType\ ID\ LPAREN\ Formals\ RPAREN\ LBRACE\ Statements\ RBRACE$
$RetType \rightarrow Type$
$RetType \rightarrow\ VOID$
$Formals \rightarrow \epsilon$
$Formals \rightarrow FormalsList$
$FormalsList \rightarrow FormalDecl$
$FormalsList \rightarrow FormalDecl\ COMMA\ FormalsList$

$FormalDecl \rightarrow Type\ ID$
$Statements \rightarrow Statement$
$Statements \rightarrow Statements\ Statement$
$Statement \rightarrow LBRACE\ Statements\ RBRACE$
$Statement \rightarrow Type\ ID\ SC$
$Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
$Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
$Statement \rightarrow Call\ SC$
$Statement \rightarrow RETURN\ SC$
$Statement \rightarrow RETURN\ Exp\ SC$
$Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
$Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
$Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
$Statement \rightarrow BREAK\ SC$
$Statement \rightarrow SWITCH\ LPAREN\ Exp\ RPAREN\ LBRACE\ CaseList\ RBRACE\ SC$
$CaseList \rightarrow CaseList\ CaseStatements$
$CaseList \rightarrow CaseStatements$
$CaseStatements \rightarrow CaseDec\ Statements$
$CaseStatements \rightarrow CaseDec$
$CaseDec \rightarrow CASE\ NUM\ COLON$
$CaseDec \rightarrow CASE\ NUM\ B\ COLON$
$CaseDec \rightarrow DEFAULT\ COLON$
$Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
$Call \rightarrow ID\ LPAREN\ RPAREN$
$ExpList \rightarrow Exp$
$ExpList \rightarrow Exp\ COMMA\ ExpList$
$Type \rightarrow INT$
$Type \rightarrow BYTE$
$Type \rightarrow BOOL$
$Exp \rightarrow LPAREN\ Exp\ RPAREN$
$Exp \rightarrow Exp\ BINOP\ Exp$
$Exp \rightarrow ID$
$Exp \rightarrow Call$
$Exp \rightarrow NUM$
$Exp \rightarrow NUM\ B$
$Exp \rightarrow STRING$
$Exp \rightarrow TRUE$
$Exp \rightarrow FALSE$
$Exp \rightarrow NOT\ Exp$
$Exp \rightarrow Exp\ AND\ Exp$
$Exp \rightarrow Exp\ OR\ Exp$
$Exp \rightarrow Exp\ RELOP\ Exp$