09.7.2017

# מבחן סוף סמסטר – מועד א'

**מרצה אחראי:** ד"ר אוהד שחם

**מתרגלים:** אבנר אליזרוב מיכל בדיאן הילה פלג

**הוראות:**

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

ב. משך המבחן שלוש שעות (180 דקות).

ג. אסור כל חומר עזר פרט לדף הנוסחאות המצורף לבחינה.

ד. במבחן 6 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

### חלק א - סיווג מאורעות (10 נקודות)

נתון קטע הקוד הבא בשפת ++C:

```
1) class A {
2) public:
3)    virtual int add(int x, int y) { return x+y; }
4)    double div(int x, int y) { return x/y; }
5)    void foo() { printf("fooA"); }
6)};
7)class B : public A {
8) public:
9)    virtual int add(int x, int y) { return x+y; }
10)   void foo(int b) { printf("fooB"); }
11)};
12)int main(){
13)   A *a = new B();
14)   B *b = new B();
15)   //… ???
16)}
```

הסעיפים הבאים מתייחסים למאורעות שיתקיימו עבור השלמות אפשריות לשורה 15. סווגו את המאורע לאחד השלבים הבאים, ונמקו: לקסיקלי, תחבירי, סמנטי ,אופטימיזציה או זמן ריצה. במידה וקיימות מספר אפשרויות, ציינו את השלב במוקדם ביותר.

1. עבור החלפת שורה 15 בפקודה הבאה, יתגלה כי יש שגיאה בתכנית:
   `a->div(3, 0);`

2. עבור החלפת שורה 15 בפקודה הבאה, יתגלה כי יש שגיאה בתכנית:
   `b->add(3 , int);`

3. עבור החלפת שורה 15 בפקודה הבאה, תיבחר הפונקציה B::foo להרצה:
   `b->foo();`

4. עבור החלפת שורה 15 בפקודה הבאה, יתגלה כי יש שגיאה בתכנית:
   `a->add(2, @);`

5. עבור החלפת שורה 15 בפקודה הבאה, תיבחר הפונקציה B::add להרצה:
   `a->add(5, 2);`

**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.

נרצה להוסיף לשפת FanC את היכולת לקבל קבועים (ליטרלים) כערכים דיפולטיים לפרמטרים בקריאות לפונקציה. כלומר, ניתן יהיה לקרוא לפונקציה עם פחות ארגומנטים מהפרמטרים המוצהרים בה, ובמקום הארגומנטים החסרים הפונקציה תשתמש בערכים שהוצהרו מראש. הצהרה על פונקציות תיראה כך:

```
int noDefaultValues(bool ok, int i, byte x) {//same as before
//statements
}
void hasDefaultValues(bool ok, int i = 6, byte x = 0b) {
//statements
}
bool allDefaultValues(bool ok = true, int i = 6, byte x = 0b) {
//statements
}
```

הערכים המושמים לפרמטר בהצהרת הפונקציה הם הערכים הדיפולטיים אשר ישמשו את הפונקציה אם פרמטר זה לא יועבר. מרגע שהוצהר פרמטר בעל ערך דיפולטי, לכל פרמטר אחריו חייב להיות ערך דיפולטי – כלומר, לא ניתן להצהיר על פונקציה כך:

```
byte thisIsWrong(bool ok = true, int i) {//…
```

בקריאה לפונקציה, חובה להעביר ארגומנטים לכל פרמטר שאין לו ערך דיפולטי, וניתן להעביר כמה ארגומנטים בעלי ערכים דיפולטיים שרוצים מתוך הפרמטרים הקיימים. אין לדלג על פרמטרים – אם הועברו k ארגומנטים לפונקציה, עליהם לתאום ל-k הפרמטרים הראשונים – כלומר, להוות רישא של רשימת הפרמטרים המלאה. כללי הטיפוסים וההמרות של FanC ישארו אותו הדבר עבור רישא זו ועבור הערכים הדיפולטיים שהוצהרו.

להלן דוגמאות לקריאות נכונות, עם הערך שיקבל כל פרמטר בריצת הפונקציה:

| | **ok** | **i** | **x** |
|---|---|---|---|
| `noDefaultValues(true,0,0b);` | true | 0 | 0b |
| `hasDefaultValues(false,0,18b);` | false | 0 | 18b |
| `hasDefaultValues(false,0);` | false | 0 | 0b |
| `hasDefaultValues(false,5b);` | false | 5 | 0b |
| `hasDefaultValues(true);` | true | 6 | 0b |
| `allDefaultValues();` | true | 6 | 0b |

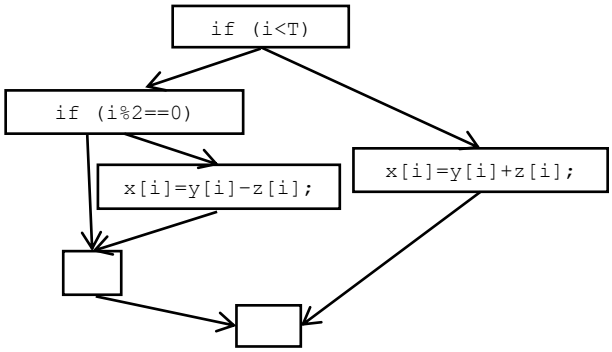
ובנוסף, דוגמאות לקריאות שגויות:

noDefaultValues(true,1);
hasDefaultValues();
allDefaultValues(8,0b);

**שאלה 2 (30 נק'): אופטימיזציות ואנליזות לויקטור**

נתון קטע הקוד הבא ו ה CFG המתאים לו :

```
if (i < T) {
    x[i] = y[i] + z[i];
} else {
    if (i%2 == 0) {
        x[i] = y[i] - z[i];
    }
}
```




כאשר T ו N קבועים. הקוד הנתון הינו בתוך לולאה ו i הינו המשתנה האינדוקטיבי של הלולאה.

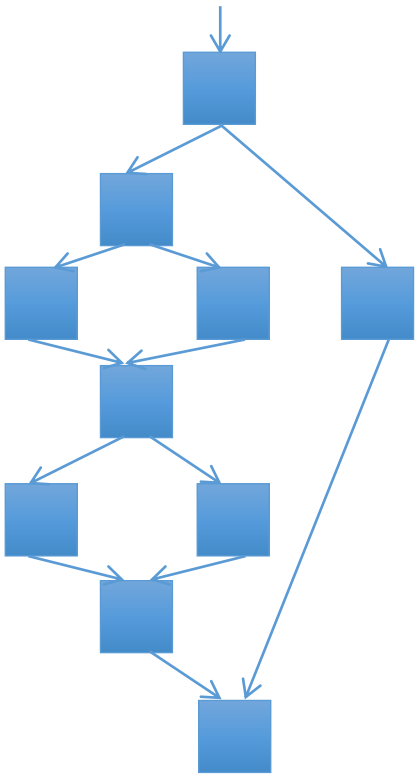
א. (2 נקודות) הסבירו אילו טרנספורמציות נדרשות על מנת לוקטר את הקוד ומדוע טרנספורמציות אלו נדרשות.

ב. (3 נקודות) הסבירו מהו החסרון העיקרי בביצוע טרנספורמציות אלו?

ג. (10 נקודות) הציגו אופטימיזציה דינמית (בזמן ריצה), המנסה להתגבר על החיסרון והסבירו איזו אנליזה נדרשת בשביל ביצוע האופטימיזציה. הסבירו עבור אילו צמתים תתבצע האופטימיזציה.

ד. (10 נקודות) הציגו אלגוריתם המבצע את האנליזה הנדרשת בסעיף 3 על ה CFG של התוכנית. היכנכם רשאים להניח שברשותכם אנליזות של dominator ו-post dominator. הסבירו עבור אלו צמתים ב CFG הייתם מבצעים את האנליזה.

ה. (5 נקודות) האם האנליזה תתן פתרון אופטימלי על ה CFG הבא? במידה ולא הרחיבו את האנליזה לקבלת פתרון אופטימלי.



**<u>שאלה 3 (15 נקודות): שאלת DFA (אנליזה סטטית) או אופימיזציה</u>**

נתון הקוד ביניים הבא, שהוא גוף של פונקציה foo

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

א. (3 נקודות) ציירו CFG לקוד הנתון כאשר כל צומת הוא בלוק בסיסי מקסימלי (לפי האלגוריתם שנלמד בכיתה).

ב. (9 נקודות) למדנו על אופטימיזציות לוקליות שניתן לבצע בתוך בלוק בסיסי, לדוגמא Constant Propagation (פעפוע קבועים). נרצה לבצע קודם בצורה גלובלית אופטימיזציה דומה שתזהה Assigned Constants - משתנים שביצעו לתוכם השמה של קבוע (שניתן לאחר מכן להחליפם בקבוע זה).

לדוגמה, הבלוק הבסיסי מסעיף א׳ שמכיל את <u>שורה 14</u> יקבל מאנליזת ה-DFA את קבוצת ה-assigned constants שמכילה רק את a שערכו 5.

הראו אנליזת DFA שמחשבת לכל בלוק בסיסי את קבוצת ה-Assigned Constants בכניסה לבלוק.

הערות:

- ניתן להגדיר שורה בודדת כבלוק בסיסי (יש לציין זאת).
- יורדו נקודות על אי <u>הגדרה מלאה</u> של DFA כפי שנלמד בתרגול. יש לציין מהם פריטי המידע, האם האנליזה היא must/may, האם היא קדמית או אחורית, ושאר פרמטרי DFA כפי שנלמדו בכיתה.
- אין צורך לממש Constant Propagation מלא.

ג. (3 נקודות) בצעו Constant Propagation **לוקלי** מספר רב של פעמים ככל האפשר עבור הבלוק הבסיסי מסעיף א׳ שמכיל את <u>שורה 14</u> (אין לבצע אופטימיזציות אחרות). היעזרו בקבוצת ה-Assigned Constants בדוגמה מסעיף ב׳ בתור המידע שהתקבל עבור הבלוק מאנליזת ה-DFA. כתבו את הקוד המעודכן של הבלוק הבסיסי לאחר ביצוע האופטימיזציה.

**שאלה 4 (10 נק'): ניתוח תחבירי וסמנטי**

נתון דקדוק SLR המגדיר רשימת מספרים, כאשר num ו-sep הם אסימונים ולאסימון num ישנה תכונה value המכילה את הלקסמה של האסימון. לדקדוק מוגדרים כללים סמנטיים שתוך כדי הניתוח במנתח SLR מדפיסים את הרשימה בסדר בו היא מופיעה בקלט, כך:

$S \rightarrow L$
$S \rightarrow \epsilon$
$L \rightarrow \underline{num}$ { println(num.value); }
$L \rightarrow L\ \underline{sep}\ \underline{num}$ {println(num.value); }

א. (5 נק') האם ניתן לגרום להדפסת הרשימה בסדר הפוך על ידי שינוי הדקדוק בלבד? אם כן, הציגו את השינוי בדקדוק והראו כי הוא SLR. אם לא, הסבירו מדוע.
ב. (5 נק') האם ניתן לגרום להדפסת הרשימה בסדר הפוך ללא כל שינוי בדקדוק? אם כן, הציגו את השינוי בכללים הסמנטיים. אם לא, הסבירו מדוע.

**שאלה 5 (10 נק'): רשומות הפעלה**

נתונה תכנית C שקומפלה עם הצהרה על הפונקציה הבאה:

```
void printParams(int x, double y);
```

כמו כן, נתון כי בהכנה לקפיצה אל פונקציה, סדר שמירת ה-frame הקודם הוא: רגיסטרים, frame pointer ישן, return address, וכן נתון כי פרמטרים לפונקציה נשמרים בסדר הפוך.

א. (4 נקודות) ציירו את רשומת ההפעלה של הקריאה `printParams(2,3)`
ב. (6 נקודות) על אף שהקוד הקורא קומפל עם הצהרה זו של הפונקציה ב-header, נפלה טעות ובתור מימוש לפונקציה קומפל המימוש הבא:

```
void printParams(int x, double y, char *z) {
     printf("%s",z);
}
```

בהנחה שהקומפילציה עברה בשלום – כלומר, הקוד הקורא מניח שהחתימה העליונה היא החתימה של הפונקציה, והקוד המבצע מניח כי קראו לו עם החתימה התחתונה – תארו מה יקרה בעת ביצוע הקריאה מסעיף א'.

## שאלה 6 (15 נקודות): Backpatching

מתכנתים רבים מעדיפים להשתמש בהדפסות ביניים כדי לדבג ולבדוק את ההתקדמות של התוכנית שכתבו בזמן ריצה.
הוצע להוסיף לשפת c את מבנה הבקרה הבא:

$S \rightarrow printf\ STRING\ every\ E_1 : L\ end$
$L \rightarrow S\ L_1 \mid S$

המבנה מאפשר להוסיף הדפסת מחרוזת, אשר תבוצע בכל שרשרת פקודות רצופה בגודל $E_1$ מתוך רשימת הפקודות $L$.
כאשר מספר הפקודות שנותרו מ L קטן או שווה ל-E נעצור את ההדפסה.
ספירת הפקודות ב L מתחילה מ1 (ולא מ0).
כלומר: אם מספר הפקודות ב-L מתחלק ב-E, לא תבוצע הדפסה בסוף L. אם מספר הפקודות ב-L קטן או שווה ל-E, לא יודפס דבר.

**דוגמת הרצה**
עבור הקוד הבא –

```
printf "-----debug-----" every 2 :
int x = 0; //s1
int y = 1; //s2
int z = x + y; //s3
printf ("%d",z); //s4
end
```

התוצאה של ההרצה היא:

```
-----debug-----
1
```

כאשר הפקודה החדשה נוספה בין הפקודה s2 ו s3.
שימו לב כי

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 לכל 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
- אין להשתמש בכללים סמנטיים באמצע כלל גזירה.
- ניתן להשתמש במרקרים N,M שנלמדו בכיתה בלבד.
- אין להשתמש במשתנים גלובליים בזמן קומפילציה.
- למשתנים S,E ישנן התכונות שהוגדרו בכיתה בלבד.
- למשתנים S,E 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 SHIFT
                else ERROR
        else    // X ∈ V
                if M[X , t] = error then ERROR
                else REPLACE(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
```

# 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` |

## 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:

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

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

## Analysis and Optimization

**Dominator:**
In control flow graph, node *D* **dominates** a node *N* if every path from the entry node to *N* must go through *D*.
**Post-dominator:**
In control flow graph, node *Z* **post dominates** a node *N* if all paths to the exit node of the graph starting at *N* must go through *Z*.

# דקדוק שפת FanC

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

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