19.02.2020

מבחן סוף סמסטר – מועד א׳

מרצה אחראי: ד״ר שחר יצחקי

מתרגלים: אדם בוטח, שקד ברודי, רפאל כהן, יעקב סוקוליק

הוראות:

א. בטופס המבחן 17 עמודים מהם 7 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 נק')

נתון קטע הקוד הבא בשפת FanC:

```
1.  enum Day {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY,
2.            FRIDAY, SATURDAY};
3.
4.  int foo9000(int i) {
5.      return i + 9000;
6.  }
7.
8.  int nextDay(enum Day today) {
9.      int tomorrow;
10.     if ((int)today == (int)SATURDAY)
11.         tomorrow = (int)SUNDAY;
12.     else
13.         tomorrow = (int)today + 1;
14.     return tomorrow;
15. }
16.
17. void main() {
18.     int x = 7;
19.     int y = 8;
20.     int z = 100 / x - y;
21.     int tomorrow = nextDay(TUESDAY);
22.     int res = foo9000(z);
23.     return;
24. }
```

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 21 בשורה הבאה:

```
21. enum Day tomorrow = (enum Day)(nextDay(TUESDAY));
```

ב. מחליפים את שורה 3 בשורה הבאה:

3. `enum WorkDay {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY};`

ג. מחליפים את שורה 18 בשורה הבאה:

18. `int x = 8;`

ד. מחליפים את שורה 7 בשורה הבאה:

7. `enum ChessPiece {PAWN, ROOK, KNIGHT, BISHOP, QUEEN, KING};`

ה. מחליפים את שורה 4 בשורה הבאה:

4. `int foo9000(int* i) {`

חלק ב – הרחבת השפה (10 נק')

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 שפת ביניים**. הקפידו על ההפרדה בין השלבים. יש להקפיד על פתרון **יעיל**.

נרצה להוסיף לשפת FanC אנוטציות מסוג Precondition. אנוטציות הינן מנגנון תחבירי נפוץ בשפות תכנות כגון Java, Python ומיועדות בדרך כלל להוספת metadata לקוד.
משמעות אנוטציות מסוג Precondition היא אכיפת תנאים מקדימים על הארגומנטים של הפונקציה.
ההגדרה של אנוטציות אלו תהיה על ידי הוספת "@pre" יחד עם התנאי המקדים שיש לאכוף, מיד לאחר חתימת הפונקציה. ניתן להשתמש ביותר מאנוטציה אחת על מנת לבדוק מספר תנאים מקדימים.
לדוגמה:

```
bool isPassing(int grade, int factor)
@pre(grade >= 0)
@pre(grade <= 100)
{
    return (grade+factor) > 55;
}
```

משמעות האנוטציות היא שבכל הפעלה של הפונקציה ייאכף התנאי `grade >= 0` ומיד לאחר מכן ייאכף התנאי `grade <= 100`.
תנאים אלו נבדקים ונאכפים בזמן ריצת התוכנית, בעת קריאה לפונקציה. במידה וכל התנאים שווים לערך true, התוכנית תמשיך לביצוע גוף הפונקציה. במידה ובקריאה לפונקציה, אחד מהתנאים המקדימים לא מתקיים, התוכנית תסיים את ריצתה.
במידה וישנם מספר תנאים מקדימים, התנאים יבדקו לפי הסדר בתוכנית המקור. בדומה ל- short-circuit evaluation, במקרה בו אחד מהתנאים המקדימים לא מסופק, התוכנית תעצור מבלי להמשיך לבדוק את התנאים הבאים.

שאלה 2 (30 נקודות): אנליזה סטטית

דונלד כתב שפת תכנות מונחית עצמים הכוללת מתודות גנריות, תוך מימוש המנגנון של template specialization (בדומה לשפת C++).

```
1   class Wall {
2     T climb<T>(int height, T rope) {
3       int j;
4       while (i < height) {
5         i = rope.jump(i, height);
6       }
7       return rope.done();
8     }
9   }
```

בשפה של דונלד יש משפטים בסיסיים מהצורה הבאה:

```
Statement → Decl | Assign | Return
Decl → Type x;
Assign → x = Expr;
Return → return Expr;
Expr → x | x.func(arg1, arg2, …) | Expr ◇ Expr
```

כאשר x ו-arg מייצגים שמות של משתנים, func מייצג שם של מתודה, ו-◇ הוא אופרטור בינארי כלשהו שעבורו טיפוסי האופרנדים הם תמיד מטיפוס int.
בנוסף קיימים משפטי בקרה מסוג if..else ו-while. התנאים במשפטי בקרה הם מטיפוס bool.

מיקי לא מרוצה מכך ששגיאות טיפוסים בגוף של מתודה גנרית יתגלו רק כאשר מקמפלים תכנית שקוראת למתודה הזו. למשל, הקוד של המחלקה Wall כמו שהוצג יתקמפל ללא שגיאות, ואולם אם המתכנת יקרא לפונקציה climb עם ערך מטיפוס שאין לו מתודה בשם jump או done תיווצר שגיאת קומפילציה בשורה 5 או 7.

א. (15 נק') תארו למיקי אנליזה סטטית שמחשבת עבור מתודה נתונה בתכנית הקלט את הממשק הנדרש מהטיפוס הגנרי T על מנת שקריאה למתודה עם טיפוס המקיים את הממשק תהיה תקינה סמנטית. למשל, עבור המתודה climb מהדוגמה, האנליזה תחזיר:

```
int jump(int, int);
T done();
```

אין צורך להתייחס למתודות של טיפוסים אחרים (למשל כאשר ערך מטיפוס T מועבר כפרמטר).

הגדירו את הדומיין האבסטרקטי, את יחס הסדר $\sqsubseteq$, ואת פונקציות המעברים עבור כל אחד מהמשפטים הבסיסיים בשפה.

הראו שפונקציות המעברים הן מונוטוניות.

אין צורך להגדיר את פונקציות האבסטרקציה (α) והקונקרטיזציה (γ).

<u>הערה</u>. **מותר** לאנליזה להשתמש במידע מטבלת הסמלים (טיפוסי הפרמטרים וטיפוס החזרה של הפונקציה). למשל, typeof(x) לציון הטיפוס של המשתנה x, ו-rtype לציון טיפוס החזרה של הפונקציה הנוכחית.

ב. (5 נק׳) בתכניות המכילות לולאות, כמה איטרציות **לכל היותר** תבצע האנליזה שנתתם למיקי על גוף הלולאה?

ג. (10 נק׳) לצורך ביצוע האנליזה היה חשוב להגדיר את המחלקה התחבירית של ביטויים (Expr) כך שארגומנט לפונקציה הוא משתנה בודד ולא ביטוי מורכב. הסבירו מדוע זה נחוץ, על-ידי שתראו תכנית שבה תכונה זו לא מתקיימת, ושעבורה לבעיה של מיקי יש **יותר מפתרון אחד אפשרי**.

שאלה 3 (20 נקודות): אופטימיזציות

להלן קטע קוד:

```
a = 1
b = 2
c = 3
e = b
d = b
f = a
while (?) {
    if (?) {
        if (?) {
            e = a
            f = a
            d = e
        } else {
            e = d
            f = c
            d = b
        }
    } else {
        if (?) {
            f = e
            e = b
            f = d
        }
    }
}
```

הסימון (?) מסמל כי התנאי לעיתים מתקיים ולעיתים לא מתקיים. שימו לב כי התנאים (?) אינם תלויים במשתנים a,b,c,d,e,f.

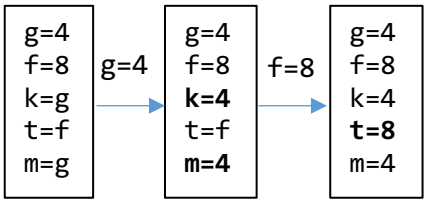
נתון כי המשתנים <u>החיים</u> לאחר קטע הקוד הם d,e,f.

א. (4 נק') ציירו את ה CFG של קטע הקוד הנתון (שאינו בשפת ביניים). הניחו כי קיים בלוק יציאה יחיד.

ב. (8 נק') בצעו אופטימיזציות **Constant Propagation**, **Useless code elimination** ככל הניתן על קטע הקוד.
בפתרונכם כתבו **אך ורק** את הקוד הסופי.
<u>הדרכה:</u> העזרו ב CFG שציירתם בסעיף הקודם.

ג. (4 נק') כעת, נניח כי בכל איטרציה של אופטמיזצית **Constant Propagation** ניתן לבצע אופטימיזציה תוך שימוש בהשמה יחידה בלבד של קבוע למשתנה.

בדוגמא המצורפת, באיטרציה הראשונה השתמשנו בהשמה g=4 בלבד , ובאיטרציה השניה השתמשנו בהשמה f=8 בלבד.



מהו מספר האיטרציות הקטן ביותר של אופטמיזצית **Constant Propagation**, עבורו יפועפעו מספר מקסימלי של קבועים בקוד המצורף לשאלה?
בתשובתכם ספקו ההשמה המתאימה לכל איטרציה.

ד. (4 נק׳) בצעו שינוי יחיד ב- right-hand-side של אחת ההשמות בקוד המקורי כך שלאחר ביצוע האופטימיזציות **Constant Propagation, Useless code elimination** (כפי שעשיתם בסעיף ב׳), מספר ההשמות מהצורה x = y (כאשר y הוא <u>משתנה</u> ו-x הוא משתנה), הוא הקטן ביותר.
לאחר השינוי, בצעו אופטימיזצית **Constant Propagation, Useless code elimination** ככל הניתן על קטע הקוד המתוקן. כתבו את השינוי שביצעתם ואת הקוד הסופי שנוצר לאחר האופטימיזציות.

שאלה 4 (15 נק'): ניתוח תחבירי וסמנטי

נתון הדקדוק הבא:

1. $S \rightarrow T\ \ C$
2. $T \rightarrow \underline{char}$
3. $T \rightarrow \underline{int}$
4. $C \rightarrow \underline{[}\ \underline{num}\ \underline{]}\ C$
5. $C \rightarrow \varepsilon$

דקדוק זה מגדיר בצורה רקורסיבית טיפוסים מסוג מערכים שגודלם ידוע מראש כאשר הטיפוס הבסיסי הוא char או int. הטרמינלים בדקדוק מסומנים בקו תחתון.

א. (5 נק') האם הדקדוק שייך למחלקה $LL(1)$?

ב. (5 נק') האם הדקדוק שייך למחלקה $LR(0)$? במידה והוא לא שייך ל-$LR(0)$, האם הוא שייך למחלקה SLR?

גודל טיפוס מסוג char הוא בית אחד וטיפוס מסוג int הוא 4 בתים. טיפוסי המערך בשפה הם בעלי מימד סופי וגודלם בזיכרון הוא מכפלת גודלי המימדים וגודל הטיפוס הבסיסי.
לדוגמה, המילה int [5][10] מגדירה טיפוס שהוא מערך דו מימדי בגודל 5*10 וגודלו בזיכרון הוא: 4*5*10.
ונניח כי קיימת מגבלת זכרון עבור טיפוס בגודל N בתים.

ג. (5 נק') נרצה לבצע בדיקה סמנטית על ביטוי בשפה (הנתונה): נרצה לוודא כי גודל הטיפוס אינו חורג ממגבלת הזכרון המקסימלית לטיפוס ובמידה וגודלו חורג להחזיר שגיאה כך:

```
throw new Error("size of array type is too large");
```

כתבו כללים סמנטיים לבדיקת התכונה הסמנטית לעיל בזמן ניתוח.
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 (15 נקודות): Backpatching

נתון מבנה הבקרה הבא:

S → every (E) run { S_List } in-reverse (B)

$S\_List \rightarrow S\ S\_List_1 \mid S$

הטרמינלים בדקדוק מסומנים בקו תחתון.

המבנה המתואר עובד באופן הבא:

- בהינתן שהערך הנגזר מהמשתנה E הינו k, והתנאי B אינו מתקיים, תתבצע כל פקודה kית ברשימת הפקודות הנגזרות ע״י S_List החל מהפקודה הראשונה (כולל) לפי הסדר.
- בהינתן שהערך הנגזר מהמשתנה E הינו k, והתנאי B מתקיים, תתבצע כל פקודה kית ברשימת הפקודות הנגזרות ע״י S_List החל מהפקודה האחרונה (כולל) בסדר **הפוך**.
- ניתן להניח כי הערך הנגזר מהמשתנה E הינו חיובי (אין צורך לבדוק זאת בקוד המיוצר).

דוגמא:

בהינתן ש-b הינו ערך בוליאני, עבור הקוד:

```
every (2) run {
    print(“I”);
    print(“me!”);
    print(“love”);
    print(“loves”);
    print(“Compi!”);
    print(“Compi”);
} in-reverse (b)
```

אם b = false תודפס התוצאה:

```
I love Compi!
```

אם b = true תודפס התוצאה:

```
Compi loves me!
```

א. (5 נק׳) הציעו פריסת קוד המתאימה לשיטת backpatching עבור מבנה הבקרה הנ״ל. על הקוד הנוצר להיות יעיל ככל האפשר.

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. כמו כן, הסבירו מהן התכונות שאתם משתמשים בהן עבור כל משתנה.

שימו לב :

- הקוד המיוצר צריך להיות בשפת הרביעיות (3 address code) כפי שנלמדה בתרגולים.
- 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, E ישנן התכונות שהוגדרו בכיתה בלבד.
- למשתנים S, B, E יש כללי גזירה פרט לאלו המוצגים בשאלה.

בהצלחה!

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

<u>הגדרה:</u> דקדוק G הוא LL(1) <u>אם ורק אם</u> לכל שני כללים ב- G השייכים לאותו משתנה A מתקיים:

$$\text{select}(A \to \alpha) \cap \text{select}(A \to \beta) = \emptyset$$

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

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

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

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

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
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 לסריקה קדמית:

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

$$\text{out}(B) = f_B\big(\text{in}(B)\big)$$

הצורה הכללית של המשוואות לסריקה אחורית:

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

$$\text{out}(B) = f_B\big(\text{in}(B)\big)$$

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:
  - place: שם המשתנה הזמני לתוכו מחושב הביטוי האריתמטי.
- M: מרקר שמטרתו שמירת כתובת פקודה. תכונות:
  - quad: כתובת הפקודה הבאה מיד לאחר המרקר.
- N: מרקר שמטרתו דילוג על קטע קוד המוסיף קוד קפיצה שיעדה עדיין לא ידוע. תכונות:
  - nextlist: רשימת כתובות המכילה את מיקום פקודת הקפיצה שנכתבה ע"י N.

שפת FanC

אסימונים:

אסימון	תבנית
VOID	void
INT	int
BYTE	byte
B	b
BOOL	bool
ENUM	enum
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	}
ASSIGN	=
RELOP	== | != | < | > | <= | >=
BINOP	+ | - | * | /
ID	[a-zA-Z][a-zA-Z0-9]*
NUM	0 | [1-9][0-9]*
STRING	״([^\n\r״\\]|\\[rnt״\\])+״

דקדוק:

1. $Program \rightarrow Enums\ Funcs$
2. $Funcs \rightarrow \epsilon$
3. $Funcs \rightarrow FuncDecl\ Funcs$
4. $FuncDecl \rightarrow RetType\ ID\ LPAREN\ Formals\ RPAREN\ LBRACE\ Statements\ RBRACE$
5. $Enums \rightarrow \epsilon$
6. $Enums \rightarrow EnumDecl\ Enums$
7. $EnumDecl \rightarrow ENUM\ ID\ LBRACE\ EnumeratorList\ RBRACE\ SC$
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. $FormalDecl \rightarrow EnumType\ ID$
16. $EnumeratorList \rightarrow Enumerator$
17. $EnumeratorList \rightarrow EnumeratorList\ COMMA\ Enumerator$
18. $Enumerator \rightarrow ID$
19. $Statements \rightarrow Statement$
20. $Statements \rightarrow Statements\ Statement$
21. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
22. $Statement \rightarrow Type\ ID\ SC$
23. $Statement \rightarrow EnumType\ ID\ SC$
24. $Statement \rightarrow EnumDecl$
25. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
26. $Statement \rightarrow EnumType\ ID\ ASSIGN\ Exp\ SC$
27. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
28. $Statement \rightarrow Call\ SC$
29. $Statement \rightarrow RETURN\ SC$
30. $Statement \rightarrow RETURN\ Exp\ SC$
31. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
32. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
33. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
34. $Statement \rightarrow BREAK\ SC$
35. $Statement \rightarrow CONTINUE\ SC$
36. $Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
37. $Call \rightarrow ID\ LPAREN\ RPAREN$
38. $ExpList \rightarrow Exp$

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