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) {
```

<u>פתרון שאלה 1א:</u>

א. שגיאה בניתוח תחבירי. בדקדוק אין כלל מתאים להמרה של טיפוסי enum.
ב. שגיאה בניתוח סמנטי. הביטוי תקין מבחינה לקסיקלית ותחבירית, אך המזהה SUNDAY כבר הוגדר לפני כן (וכך גם שאר הימים המופיעים בשורה החדשה).
ג. אין שגיאה. נשים לב כי הביטוי בשורה 20 שקול ל 8 – 8 / 100, ולפי עדיפויות האופרטורים שהוגדרו עבור FanC, קודם יתבצע החילוק ולאחר מכן החיסור – כך שאין חלוקה ב-0.
ד. שגיאה בניתוח התחבירי. נשים לב שלפי הכלל הראשון בשפת FanC, כל הגדרות טיפוסי ה-enum הגלובליים צריכים להיכתב לפני הפונקציות. לכן לא ניתן להגדיר enum לאחר הגדרת פונקציה כלשהי.
ה. שגיאה בניתוח התחבירי. int* מתורגם, לאחר הניתוח הלקסיקלי, לרצף האסימונים <INT><BINOP>, ולא מתאים לאף כלל בשפה.

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.

פתרון שאלה 1ב:

- ניתוח לקסיקלי: נגדיר אסימון חדש, PRECOND, אשר יתאים לתבנית @pre.
- ניתוח תחבירי: נוסיף לדקדוק את הכללים הבאים, אשר גוזרים רשימת PreConditions:

PreConditions -> ε
PreConditions -> PreConditions PreCondition
PreCondition -> PRECOND LPAREN Exp RPAREN

ונשנה את הכלל המתאים להצהרת פונקציות בצורה הבאה:

FuncDecl -> RetType ID LPAREN Formals RPAREN PreConditions LBRACE Statements RBRACE

- ניתוח סמנטי: נוודא כי Exp הנגזר בכלל המתאים ל-PreCondition הינו מטיפוס בוליאני. נשים לב כי בדיקות סמנטיות אחרות (כמו למשל האם משתנה המופיע בתנאי מסוים הוא אכן ארגומנט של הפונקציה) יקרו באופן טבעי. זאת כיוון שבזמן גזירת PreConditions, טבלת הסמלים כבר מכילה את הארגומנטים של הפונקציה.
- ייצור קוד: נתרגם את רשימת ה-PreConditions למכפלת AND בין כל הביטויים (לפי הסדר), ולאחר מכן נבצע backpatching של ה-truelist של ה-AND אל גוף הפונקציה, ושל ה-falselist שלו אל קטע קוד אשר יסיים את ריצת התוכנית.

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**.

פתרון שאלה 2:

א.

הדומיין יהיה סריג קבוצת חזקה מעל קבוצת כל החתימות האפשריות של מתודות בשפה. חתימה היא שלשה מהצורה (שם, טיפוס חזרה, רשימת טיפוסי הפרמטרים).

$L = P(Ids \times Types \times Types^*)$

יחס הסדר בסריג הוא הכלת קבוצות.

פונקציות המעבר — סמנטיקה אבסטרקטית של משפטים בסיסיים:

$[\![x = y.func(arg_1, arg_2, \ldots)]\!]^\# \sigma^\# = \sigma^\# \cup \{\langle \text{"func"}, typeof(x), \langle typeof(arg_1), typeof(arg_2), \ldots \rangle\rangle\}$
$[\![return\ y.func(arg_1, arg_2, \ldots)]\!]^\# \sigma^\# = \sigma^\# \cup \{\langle \text{"func"}, rtype, \langle typeof(arg_1), typeof(arg_2), \ldots \rangle\rangle\}$
$[\![if\ y.func(arg_1, arg_2, \ldots)]\!]^\# \sigma^\# = \sigma^\# \cup \{\langle \text{"func"}, bool, \langle typeof(arg_1), typeof(arg_2), \ldots \rangle\rangle\}$
$[\![x = e_1 \diamond e_2]\!] \sigma^\# = \sigma^\# \cup [\![e_1]\!]^\# \cup [\![e_2]\!]^\#$
$[\![return\ e_1 \diamond e_2]\!] \sigma = \sigma^\# \cup [\![e_1]\!]^\# \cup [\![e_2]\!]^\#$
$[\![if\ e_1 \diamond e_2]\!]^\# \sigma^\# = \sigma^\# \cup [\![e_1]\!]^\# \cup [\![e_2]\!]^\#$

סמנטיקה אבסטרקטית של ביטויים (שימו לב שהיא מופעלת רק כאשר משפט מכיל ביטוי שיש בו אופרטור):

$[\![y.func(arg_1, arg_2, \ldots)]\!]^\# = \{\langle \text{"func"}, int, \langle typeof(arg_1), typeof(arg_2), \ldots \rangle\rangle\}$
$[\![e_1 \diamond e_2]\!]^\# = [\![e_1]\!]^\# \cup [\![e_2]\!]^\#$
$[\![x]\!]^\# = \emptyset$

כאשר הטיפוס של y הוא T (הטיפוס הגנרי), הטיפוס של x הוא typeof(x) וכן לגבי $typeof(arg_i)$.

עבור כל שאר המשפטים בשפה (שאינם מכילים קריאה למתודה של הטיפוס הגנרי) הפונקציה היא זהות.

$[\![s]\!]^\# \sigma^\# = \sigma^\#$

ב.

לכל היותר פעמיים, זאת מכיוון שהאנליזה רק מוסיפה איברים ל $\sigma^\#$ ואיברים אלה אינם תלוים ב $\sigma^\#$, לכן במעבר השני כל האיברים כבר נמצאים והמצב האבסטרקטי יהיה שווה למצב באיטרציה הקודמת.

ג.

נשנה את התוכנית לדוגמה כך:

```
class Wall {
```

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

כעת אין תשובה חד משמעית למה צריך להיות ערך החזרה של T.get. כל פתרון שבו טיפוס החזרה של T.get שווה לטיפוס הפרמטר השני של T.jump הוא פתרון 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.

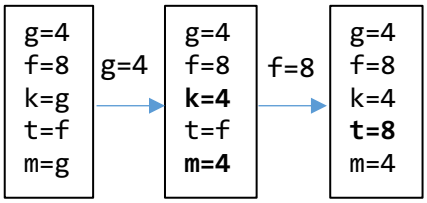
נתון כי המשתנים <u>החיים</u> לאחר קטע הקוד הם d,e,f.

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.

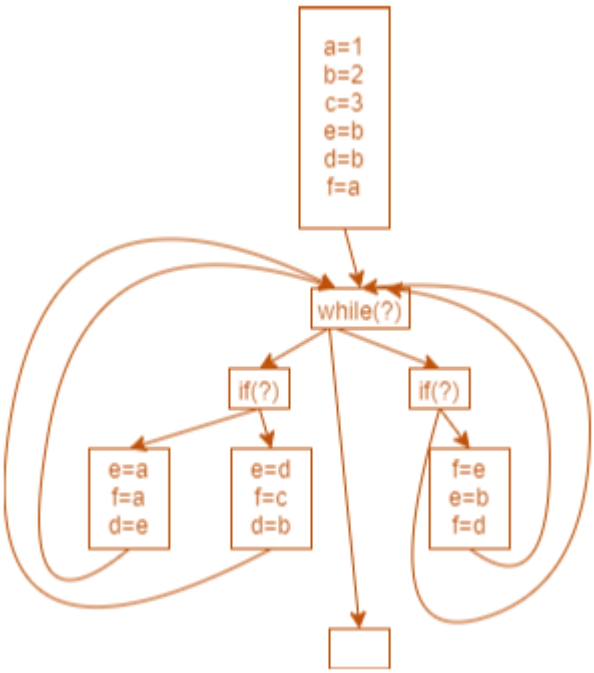


מהו מספר האיטרציות הקטן ביותר של אופטמיזצית **Constant Propagation**, עבורו יפועפעו מספר מקסימלי של קבועים בקוד המצורף לשאלה?
בתשובתכם ספקו ההשמה המתאימה לכל איטרציה.

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.

פתרון שאלה 3:

א.



ב.

```
1.  e = 2
2.  d = 2
3.  f = 1
4.  while (?) {
5.      if (?) {
6.          if (?) {
7.              e = 1
8.              f = 1
9.              d = 1
10.         } else {
11.             e = d
12.             f = 3
13.             d = 2
14.         }
15.     } else {
16.         if (?) {
17.             e = 2
18.             f = d
19.         }
20.     }
21. }
```

נשים לב שהערכים של a,b,c לא משתנים במהלך התוכנית.

בנוסף, בבלוק של ה if הפנימי הראשון, ניתן לפעפע את הערך של e לתוך d, מכיוון שהוא נשאר קבוע לאורך כל התוכנית.

ג. נבצע אופטימזציה בסדר הבא:

a->b->c->e

למעשה, כל פרמוטציה של הארבעה, כך שהאופטימזציה של a תבוא לפני האופטימיזציה של e תהיה נכונה.

ד. נחליף את ההשמה בשורה 10 ל- e = b ונקבל את הקוד הבא לאחר האופטימיזציה:

```
1.  e = 2
2.  d = 2
3.  f = 1
4.  while (?) {
5.      if (?) {
6.          if (?) {
7.              e = 2
8.              f = 1
9.              d = 2
10.         } else {
11.             e = 2
12.             f = 3
13.             d = 2
14.         }
15.     } else {
16.         if (?) {
17.             e = 2
18.             f = 2
19.         }
20.     }
21. }
```

נשים לב כי כעת בתוך הלולאה הערכים של d , e לא משתנים ולכן ניתן לבצע עליהם אופטימיזצית **Constant Propagation** במקומות נוספים.

הערה: בסוף הקוד המצורף , בהכרח d=2,e=2. לכן, ההשמות האלו בתוך ה while מיותרות.

באנליזה שראינו בשיעור להסרת dead code, לא ניתן להבין כי ניתן למחוק את שורות אלו, אך התשובה הבאה גם תתקבל:

```
1.  e = 2
2.  d = 2
3.  f = 1
4.  while (?) {
5.      if (?) {
6.          if (?) {
7.              f = 1
8.          } else {
9.              f = 3
10.         }
11.     } else {
12.         if (?) {
13.             f = 2
14.         }
15.     }
16. }
```

יתכנו פתרונות נוספים כמו לשנות את שורה 12 ל d=2

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:

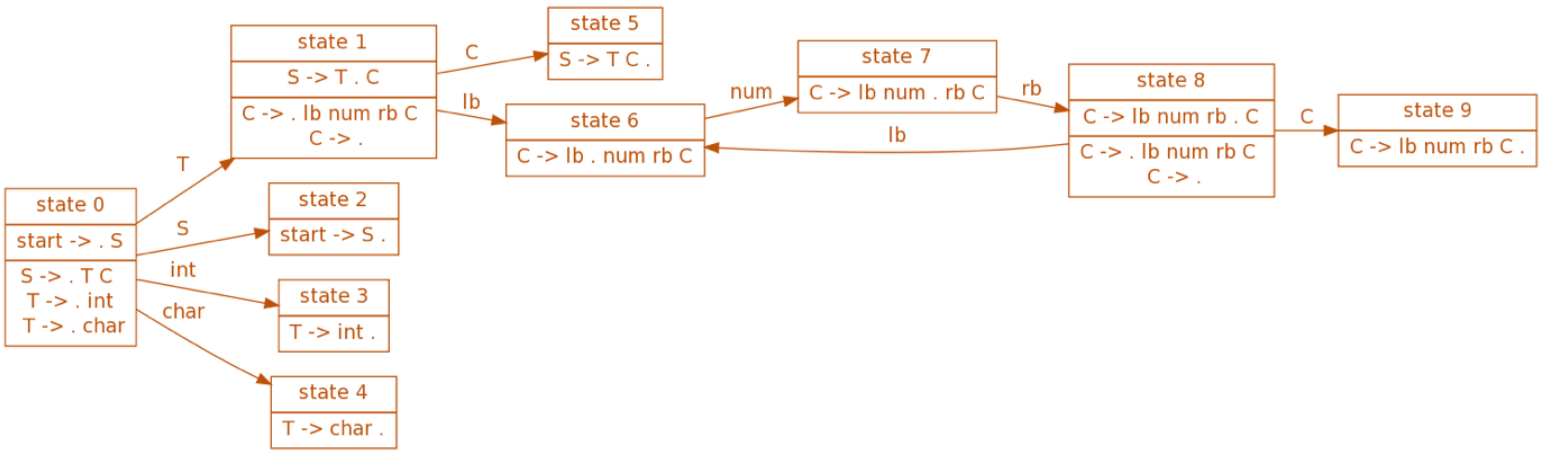
- אין לשנות את הדקדוק.
- יש לבצע את הניתוח הסמנטי בזמן בניית עץ הגזירה.
- ניתן להוסיף למשתנים תכונות סמנטיות כרצונכם. יש לציין אותן מפורשות.
- אין להשתמש במשתנים גלובליים.
- יש לכתוב את הכללים הסמנטיים במלואם.

פתרון שאלה 4:

א. הדקדוק שייך ל-$LL(1)$.

	int	char	[	num	]	$
S	$S \to T\ C$	$S \to T\ C$				
T	$T \to int$	$T \to char$				
C			$C \to [\ num\ ]\ C$			$C \to \varepsilon$

ב. הדקדוק אינו שייך ל-$LR(0)$ אך שייך ל-SLR.



ג.

```
S → T C {
    if (T.size * C.size  > N)
        throw new Error("size of array type is too large");
}

T → int {
    T.size = 4;
}

T → char {
    T.size = 1;
}

C → [ num ] C1 {
    C.size = C1.size * num;
}
C → ε {
    C.size = 1;
}
```

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!

פתרון שאלה 5:

נוסיף מרקרים במקומות המסומנים:

$S \rightarrow$ **every** ($E\ N$) **run** { S_List } **in-reverse** ($M\ B$)
$S_{List} \rightarrow M\ S\ S\_List_1 \mid M\ S$

נוסיף את התכונות הסמנטיות הבאות:

S_List: quad_list, nextlist

פריסת הקוד:

S

$E \rightarrow t_1$

$goto\ \ check\_reversed$

S_1 //nextlist: Inc

S_2 //nextlist: Inc

.
.
.

S_n //nextlist: Inc

$\underline{check\ reversed}$:

B //truelist: rev
//falselist: not_rev

$\underline{rev}$:
$t_3 = n$
$t_1 = 0 - t_1$
$goto\ \underline{compare}$
$\underline{not\ rev}$:
$t_3 = 1$
$goto\ \underline{compare}$
$\underline{inc}$:
$t_3 = t_3 + t_1$
$\underline{compare}$:
$If\ t_3 == 1\ \ goto\ S_1$
$If\ t_3 == 2\ \ goto\ S_2$
.
.
.
$If\ t_3 == n\ \ goto\ S_n$
$goto$ _____ $(next)$

סכימת התרגום:

```
S_List → M S {
S_List.quad_list = newstack();
S_List.quad_list.push(M.quad);
S_List.nextlist = S.nextlist;
}

S_List → M S S_List1 {
S_List.quad_list = S_List1.quad_list;
S_List.quad_list.push(M.quad);
S_List.nextlist = merge(S.nextlist, L1.nextlist);
}

S → every ( E N ) run { S_List } in-reverse ( M B ) {
backpatch(N.nextlist, M.quad);
backpatch(B.truelist, nextquad());
curr_index = newtemp(); // t3
emit(curr_index || " = " || S_List.quad_list.size());
emit(E.place || " = 0 - " || E.place);
compare_list = makelist(nextquad());
emit("goto ");
backpatch(B.falselist, nextquad());
emit(curr_index || " = 1");
compare_list = merge(compare_list, makelist(nextquad()));
emit("goto ");
backpatch(S_List.nextlist, nextquad());
emit(curr_index || " = " || curr_index || " + " || E.place );
backpatch(compare_list, nextquad());
i = 1;
while (!S_List.quad_list.empty()) {
        quad = S_List.quad_list.pop();
        emit("if " || curr_index || " == " || i || "goto " || quad);
        i++;
}
S.nextlist = makelist(nextquad());
emit("goto ");
}
```