15.09.2019

מבחן סוף סמסטר – מועד ב'

מרצה אחראי: ד"ר בלהה מנדלסון

מתרגלים: אנטוניו אבו-נסאר, שקד ברודי, יעקב סוקוליק, יואב צוריאל

הוראות:

א. בטופס המבחן 14 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 נק')

נתון קטע הקוד הבא בשפת FanC:

```
1.  int remainder(int x, int y)
2.  @pre(y != 0)
3.  {
4.      return x - y * (x / y);
5.  }
6.
7.  bool isPrime(int x)
8.  {
9.      int i = 2;
10.     bool flag = false;
11.     while(i <= x / 2)
12.     {
13.         if (remainder(x, i) == 0)
14.         {
15.             flag = true;
16.             break;
17.         }
18.         i = i + 1;
19.     }
20.     if (x == 1)
21.         return false;
22.     else
23.         return not flag;
24. }
25.
26. void main()
27. {
28.     if (isPrime(42))
29.         print("42 is prime");
30. }
```

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 2 בשורה הבאה:

```
2. @pre()
```

ב. מוחקים את שורה 10.

ג. מחליפים את שורה 23 בשורה הבאה:

```
23.         return !flag;
```

ד. מחליפים את שורה 28 בשורה הבאה:

```
28.     if (isPrime(42b))
```

ה. מחליפים את שורה 29 בשורה הבאה:

```
29.         printi("42");
```

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

נרצה להוסיף לשפת FanC משתנים סטטיים בדומה לשפת C. משתנה סטטי אינו משוחרר בעת יציאה מה-scope שבו הוא הוגדר, ומשמר את ערכו בין כניסות שונות ל-scope זה למשך כל ריצת התוכנית, בשונה ממשתנה לוקאלי שמשוחרר ביציאה מה-scope שבו הוא הוגדר ובעת כניסה נוספת ל-scope הוא יוקצה מחדש.
בכדי להגדיר משתנה כמשתנה סטטי נדרש להוסיף לפני הגדרת טיפוס המשתנה את המילה השמורה static.
משתנה סטטי מאותחל רק פעם אחת בזמן ריצת התוכנית והוא יכול להיות מאותחל רק עם ערך קבוע.
ניתן לאתחל משתנה סטטי באופן מפורש עם ערך, לדוגמה:

```
static int i = 42;
```

או באופן לא מפורש ואז הוא יאותחל כברירת מחדל לערך 0, לדוגמה:

```
static int i; // i is initialized with 0
```

לדוגמה, הרצת התוכנית:

```
int fun()
{
    static int count = 0;
    count = count + 1;
    return count;
}

void main()
{
    printi(fun());
    printi(fun());
}
```

תדפיס את הערכים:

```
1
2
```

שאלה 2 (20 נקודות): אנליזה סטטית

```
1.  a = 1
2.  b = 2
3.  c = a + b
4.  d = c - a
5.  d = b + d
6.  if d > 0 goto 10
7.  d = a + b
8.  e = e + 1
9.  goto 5
10. b = a + b
11. e = c - a
12. if e < 0 goto 3
13. a = b * d
14. b = a - d
```

א. (8 נק׳) בנו את גרף הזרימה (control flow graph) עבור קטע קוד הנ״ל.

ב. (12 נק׳) עבור הגרף הזרימה (control flow graph) שבניתם, חשבו את הקבוצות gen, kill, IN ו-OUT עבור ניתוח חיות (live variable analysis) של המשתנים בקוד והראו את דרך החישוב.
הניחו שכל המשתנים חיים בסוף התכנית.

שאלה 3 (20 נקודות): אופטימיזציות

א. (4 נק') הסבירו מהו מנגנון ה-JIT (Just In Time).

אליוט, עובד בחברת Oopsi (**O**ptimizations f**O**r **P**recompiled **S**oftware **I**nc.) התבקש לתכנן מנגנון אופטימיזציה חדש עבור שפת תכנות שהיא interpreted (המשתמשת במפרש בזמן ריצה) ולא משתמשים בה במנגנון ה-JIT.
הוא הציע את הרעיון הבא:

- נגדיר מילה שמורה חדשה: "@opt".
- נשתמש באופטימיזציה, אך ורק עבור פונקציות שסומנו ע"י המילה השמורה, למשל:

```
@opt
void foo(int x) {
     return x + 1;
}
```

- המנגנון יעבוד כך: בפעם הראשונה שנכנסים לפונקציה שסומנה עם @opt, המפרש (interpreter) ישמור את כל פקודות המכונה שהוא פירש במהלך ביצוע הפונקציה.

בפעמים הבאות שהפונקציה תקרא, במקום לתרגם כל פקודה, נשתמש ברצף הפקודות ששמרנו בריצה הראשונה.
שימו לב, במידה ובמהלך הקריאה הראשונה לפונקציה שסומנה עם "@opt", הפונקציה שלנו נקראה שוב (ע"י רקורסיה למשל) לא נשמור שוב את רצף הפקודות שכבר תורגמו, אלא נשמור פקודת קריאה לפונקציה שעושים עבורה אופטימיזציה. המפרש ימשיך לתרגם את הפקודות לביצוע (שבקריאה הנוספת) לשפת מכונה ויבצע אותן (כאמור, מבלי לשמור אותן). ברגע שנחזור חזרה לקריאה הראשונה, המנגנון ימשיך לשמור את פקודות המכונה.

הבוס של אליוט בחן את הרעיון ודחה אותו בטענה כי נכונות התוכנית עלולה להיפגע.

ב. (3 נק') תארו את המקרים בהם נכונות התוכנית עלולה להיפגע. הסבירו את תשובתכם.

כעת, אליוט רוצה לשפר את המנגנון שלו כך שנכונות התוכנית לא תפגע.

ג. (6 נק') עזרו לאליוט והציעו בקצרה אלגוריתם שידע להגיד עבור כל פונקציה האם ניתן לסמן אותה עם "@opt" או לא. שימו לב, פונקציה יכולה להיות מסומנת עם "@opt" אמ"מ הנכונות שלה לא תפגע.

לאחר השינויים, אליוט הציג את הרעיון לבוס שלו, אך הוא שוב דחה אותו מכיוון שישנן אופטימיזציות נוספות שלא ניתן להפעיל עם המנגנון.

הניחו כי לא ניתן לבצע אופטימיזציות על פקודות מכונה לאחר התרגום. עם זאת, ניתן לבצע אופטימיזציות על הפקודה בשפה העילית לפני התרגום.

ד. (3 נק') הסבירו למה הבוס של אליוט התכוון, ופרטו אילו אופטימיזציות לא ניתן להפעיל עם המנגנון של אליוט.

בנוסף, הבוס של אליוט טוען כי בניגוד למנגנון JIT בשפת Java, המאפשר איזון בין הנוחות של המפרש (Interpreter) ליעילות של קוד מקומפל, המנגנון של אליוט אינו מציע איזון זה.

ה. (4 נק') הסבירו למה הבוס של אליוט התכוון ופרטו כיצד המנגנון של אליוט פוגע באיזון זה. בפרט, התייחסו כיצד JIT משיג את האיזון הזה.

<u>שאלה 4 (20 נק'): ניתוח תחבירי וסמנטי</u>

מצאו דקדוק G במחלקה $LL(1)$ שאינו נמצא במחלקה SLR.

בעת בניית הדקדוק, כל משתנה X בדקדוק חייב להיות ישיג מהמשתנה ההתחלתי S – כלומר קיימת סדרת גזירות המתחילה ב-S ומגיעה לתבנית פסוקית שבה מופיע המשתנה.

א. (10 נק') הוכיחו כי G שמצאתם שייך ל- $LL(1)$.

ב. (10 נק') הראו כי G שמצאתם אינו שייך ל-SLR. מצאו קונפליקט באוטומט, והראו אותו בטבלת הניתוח.

שאלה 5 (20 נקודות): ייצור קוד

א. (6 נק') תנו יתרון אחד וחסרון אחד להעברת פרמטרים רק דרך רגיסטרים.

ב. (14 נק') לפניכם הקוד הבא בשפת FanC:

```
a = x + y;
d = 8;
c = 3 * (a + d);
if (a >= d or c < a)
     print("True!");
else
     print("False...");
print("Done.");
```

תרגמו אותו לקוד בשפת הרביעיות שראיתם בתרגול (התחלנו עבורכם את התרגום).

אין לאחסן תוצאות של ביטויים בוליאניים בתוך משתנים זמניים.

הניחו כי כל משתנה זמני יכול לקבל ערך פעם אחת בלבד.

ניתן להשתמש במשתנים רק בפעולות קריאה (שורה 1 בפתרון) או כתיבה (שורה 4 בפתרון) ולא בפעולות של חישוב או תנאי. בפעולות אלו יש להשתמש **אך ורק** במשתנים זמניים.

אין צורך לכתוב קוד מיוחד עבור קריאה לפונקציה, לדוגמה ניתן לכתוב בקוד את הפקודה: `print("True!")`.

```
1.  t1 = x
2.  t2 = y
3.  t3 = t1 + t2
4.  a = t3
...
```

בהצלחה!

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

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

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

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

<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
```

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
AND	and
OR	or
NOT	not
TRUE	true
FALSE	false
RETURN	return
IF	if
ELSE	else
WHILE	while
BREAK	break
CONTINUE	continue
PRECOND	@pre
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. $Program \rightarrow Funcs$
2. $Funcs \rightarrow \epsilon$
3. $Funcs \rightarrow FuncDecl\ Funcs$
4. $FuncDecl \rightarrow$
   $RetType\ ID\ LPAREN\ Formals\ RPAREN\ PreConditions\ LBRACE\ Statements\ RBRACE$
5. $RetType \rightarrow Type$
6. $RetType \rightarrow\ VOID$
7. $Formals \rightarrow \epsilon$
8. $Formals \rightarrow FormalsList$
9. $FormalsList \rightarrow FormalDecl$
10. $FormalsList \rightarrow FormalDecl\ COMMA\ FormalsList$
11. $FormalDecl \rightarrow Type\ ID$
12. $PreConditions \rightarrow \epsilon$
13. $PreConditions \rightarrow PreConditions\ PreCondition$
14. $PreCondition \rightarrow PRECOND\ LPAREN\ Exp\ RPRAEN$
15. $Statements \rightarrow Statement$
16. $Statements \rightarrow Statements\ Statement$
17. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
18. $Statement \rightarrow Type\ ID\ SC$
19. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
20. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
21. $Statement \rightarrow Call\ SC$
22. $Statement \rightarrow RETURN\ SC$
23. $Statement \rightarrow RETURN\ Exp\ SC$
24. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
25. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
26. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
27. $Statement \rightarrow BREAK\ SC$
28. $Statement \rightarrow CONTINUE\ SC$
29. $Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
30. $Call \rightarrow ID\ LPAREN\ RPAREN$
31. $ExpList \rightarrow Exp$
32. $ExpList \rightarrow Exp\ COMMA\ ExpList$
33. $Type \rightarrow INT$
34. $Type \rightarrow BYTE$
35. $Type \rightarrow BOOL$
36. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
37. $Exp \rightarrow Exp\ BINOP\ Exp$

38. $Exp \rightarrow ID$
39. $Exp \rightarrow Call$
40. $Exp \rightarrow NUM$
41. $Exp \rightarrow NUM\ B$
42. $Exp \rightarrow STRING$
43. $Exp \rightarrow TRUE$
44. $Exp \rightarrow FALSE$
45. $Exp \rightarrow NOT\ Exp$
46. $Exp \rightarrow Exp\ AND\ Exp$
47. $Exp \rightarrow Exp\ OR\ Exp$
48. $Exp \rightarrow Exp\ RELOP\ Exp$