מבחן סוף סמסטר – מועד א'

טור מקור

מרצה אחראי: ד"ר הילה פלג

מתרגלים: הילה לוי, תומר כהן, אנדריי בבין, תומר ביתן

הוראות:

1. בטופס המבחן 14 עמודים, מתוכם 5 דפי נוסחאות. בדקו שכל העמודים ברשותכם.
2. משך המבחן שלוש שעות (180 דקות).
3. כל חומר עזר חיצוני אסור לשימוש.
4. בשאלות הפתוחות, ניתן לציין לגבי סעיף או שאלה "לא יודע/ת". תשובה זו תזכה ב-20% מהניקוד. תשובות שגויות לא יזכו בניקוד.
5. קראו את כל המבחן לפני שאתם מתחילים לענות על השאלות.
6. אין צורך להגיש את טופס מבחן זה בתום הבחינה.
7. את התשובות לשאלות הסגורות יש לסמן בטופס הטורי הנפרד בלבד ואת התשובות לשאלות הפתוחות יש לכתוב בריבועים הריקים לכל שאלה בטופס התשובות הנפרד.
8. ודאו כי אתם מגישים טופס תשובות וטופס טורי בלבד (את מחברת הטיוטה ניתן לשמור אצלכם).

בהצלחה!

חלק א' - שאלות סגורות (45 נק')

שלבי קומפילציה (27 נק')

נתונה התוכנית הבאה בשפת FanC:

```
1.  int x = 3;
2.  int y = 5;
3.  int m = x - y;
4.  int n = x * y;
5.
6.  int k = 0;
7.  while (k * k < n) {
8.      k = k + n / m;
9.  }
10.
11. print("Result:");
12. printi(k);
```

בסעיפים הבאים (שאלות 1 עד 5) מוצגים שינויים (בלתי תלויים) לקוד של התוכנית. עבור כל שינוי ציינו את השלב המוקדם ביותר שבו נגלה את השגיאה:

שאלה מספר 1:

(3 נק') מחליפים את שורה 7 ב-

```
while (k ^ 2 < n)
```

א. שגיאה בניתוח לקסיקלי

ב. שגיאה בניתוח הסמנטי

ג. שגיאה בניתוח תחבירי

ד. אין שגיאה

ה. שגיאה בייצור קוד

ו. שגיאה בזמן ריצה

שאלה מספר 2:

(3 נק') מחליפים את שורה 6 ב-

```
byte k = 0b;
```

א. שגיאה בניתוח סמנטי

ב. שגיאה בניתוח תחבירי

ג. אין שגיאה

ד. שגיאה בניתוח לקסיקלי

ה. שגיאה בייצור קוד

ו. שגיאה בזמן ריצה

שאלה מספר 3:
(3 נק') מחליפים את שורה 2 ב-

```
int y = -5;
```

א. שגיאה בניתוח תחבירי
ב. אין שגיאה
ג. שגיאה בניתוח לקסיקלי
ד. שגיאה בניתוח סמנטי
ה. שגיאה בייצור קוד
ו. שגיאה בזמן ריצה

שאלה מספר 4:
(3 נק') מחליפים את שורה 2 ב-

```
int y = 3;
```

א. שגיאה בזמן ריצה
ב. שגיאה בניתוח סמנטי
ג. אין שגיאה
ד. שגיאה בניתוח לקסיקלי
ה. שגיאה בניתוח תחבירי
ו. שגיאה בייצור קוד

שאלה מספר 5:
(3 נק') מחליפים את שורה 11 ב-

```
string s = “Result:”;
print(s);
```

א. שגיאה בניתוח תחבירי
ב. שגיאה בניתוח לקסיקלי
ג. אין שגיאה
ד. שגיאה בניתוח סמנטי
ה. שגיאה בייצור קוד
ו. שגיאה בזמן ריצה

שינויים ב-FanC:

שאלה מספר 6:
(6 נק') נרצה להוסיף לFanC משפט postfix increment כמו ++ של C, אבל הכמות בה המשתנה מוגדל מוגדרת לפי מספר הפלוסים המופיעים ברצף.
למשל:

```
x++;
```

יגדיל את x ב-1. ו:

```
y++++;
```

יגדיל את y ב-3.

מהו שלב הקומפילציה **המוקדם** ביותר שצריך לשנות על מנת לתמוך במשפט החדש?

א. ניתוח לקסיקלי
ב. ייצור קוד
ג. ניתוח סמנטי
ד. ניתוח תחבירי
ה. לא נצטרך לשנות אף אחד מהשלבים

שאלה מספר 7:
(6 נק') לאחר שמימשנו את המשפט החדש, נרצה להפוך את האופרטור ממשפט לביטוי.
מהו שלב הקומפילציה **המאוחר** ביותר **שלא** צריך לשנות מפתרון השאלה הקודמת?

א. ניתוח לקסיקלי
ב. לא נצטרך לשנות אף אחד מהשלבים
ג. ניתוח סמנטי
ד. ניתוח תחבירי
ה. ייצור קוד

אופטימיזציות (18 נק')

שאלה מספר 8:
נתונה התוכנית הבאה בשפת הרביעיות:

```
1.  a = 1
2.  x = a
3.  if x > 5 goto 6
4.  x = x + 1
5.  goto 3
6.  c = 3
7.  if x > 10 goto 13
8.  x = x + 1
9.  goto 10
10. goto 12
11. x = x - 1
12. goto 3
13. print c
```

(6 נק') הריצו על המתודה את האופטימיזציה branch chaining עד להתכנסות, כמה בלוקים וכמה קשתות יש בגרף?

א. 8 בלוקים, 9 קשתות

ב. 8 בלוקים, 8 קשתות

ג. 9 בלוקים, 9 קשתות

ד. 7 בלוקים, 7 קשתות

ה. 7 בלוקים, 8 קשתות

שאלה מספר 9:
(6 נק') נתונה גרסת אינטרפרטר (מפרש) של FanC. מתי משתלם להשתמש בגרסה זו על פני הגרסה של FanC מתרגילי הבית שלכם? ענו את התשובה הנכונה ביותר.

א. עדיף להשתמש בקומפיילר אם הולכים להריץ את התוכנית יותר מפעם אחת.

ב. תמיד עדיף להשתמש בגרסת אינטרפרטר.

ג. תמיד עדיף להשתמש בקומפיילר.

ד. עדיף להשתמש באינטרפרטר אם הולכים להריץ את התוכנית יותר מפעם אחת.

שאלה מספר 10:
(6 נק') נוסיף לקומפיילר אופטימיזציות ולאינטרפרטר (מפרש) JIT.
נתונה תוכנית אשר אנחנו מתכננים להריץ פעם אחת בלבד.
מתי עדיף להשתמש באינטרפרטר החדש ולא בקומפיילר החדש על התוכנית הנתונה?

א. עדיף להשתמש בגרסת האינטרפרטר כשהתוכנית קטנה וכוללת קטע קוד שירוץ מספר גדול מאוד של פעמים.

ב. עדיף להשתמש בגרסת האינטרפרטר כשהתוכנית גדולה מאוד ותרוץ בזמן רב.

ג. להרצה אחת תמיד עדיף להשתמש בגרסת אינטרפרטר.

ד. להרצה אחת תמיד עדיף להשתמש בקומפיילר.

ה. אף פעם אי אפשר להניח שהתוכנית תרוץ רק פעם אחת.

חלק ב' - שאלות פתוחות (55 נק')

שאלה 1: דקדוקים (25 נק')

א. (8 נק') נתון הדקדוק הבא G_1 :

$$E \rightarrow lpar\ Binop\ rpar \mid id \mid num$$
$$Binop \rightarrow\ add\ E\ E \mid sub\ E\ E$$

כאשר $id, num, lpar, rpar, add, sub$ הינם אסימונים. האם הדקדוק ב LL(1)? נמקו.

ב. (8 נק') הוחלט להרחיב את השפה מעט, ולשם כך גם הוחלפו הפעולות הבינאריות מ-prefix notation ל- infix notation. הדקדוק החדש G_2 הוא:

$$S \rightarrow E\ sc\ S \mid E\ sc$$
$$E \rightarrow E\ sub\ E$$
$$E \rightarrow E\ add\ E$$
$$E \rightarrow id$$
$$E \rightarrow num$$

כאשר id, num, sc הינם אסימונים זהים לאסימונים בתרגילי הבית, add הוא הסימן + ו-sub הוא הסימן -. הדקדוק גוזר לדוגמה את רצף האסימונים מהמילים ω_1, ω_2:

$\omega_1 =$ `2 + 3 + x; x + y - x + 1; 1`
$\omega_2 =$ `2`

ציירו את אוטומט LR(0) המלא של G_2, והראו שהדקדוק אינו ב- $LR(0)$. יש לסמן את כל הקונפליקטים.

ג. (2 נק') ניתן לפתור את הקונפליקטים על ידי הפעלת מנגנון פתרון הקונפליקטים של bison על אוטומט LR(0). הניחו שכרגע כל האסימונים מוגדרים `%token`, וציינו אילו סדר ואסוציאטיביות יפתרו את הקונפליקטים – והסבירו כיצד.

ד. (7 נק') נגדיר על סדרת ביטויים בשפה של הדקדוק G_2 את התכונה "מתגלה לאיטו": רצף ביטויים מתגלה לאיטו אם בכל ביטוי יש **בדיוק** את כל המשתנים מהביטוי משמאלו ועוד משתנה אחד חדש.

לדוגמה:

$\omega_3 =$ `x + y; 1; 3 + y;`
$\omega_4 =$ `x; x + y; x - 2 + y - z;`
$\omega_5 =$ `x + y; x - y + x - z; x + y - z + w;`
$\omega_6 =$ `2; 1 - x;`

ω_3 לא מתגלה לאיטו, משום שבביטוי השני לא מופיעים משתנים כלל, אך לפי ההגדרה גם x, גם y וגם משתנה נוסף היו צריכים להופיע בו. לעומת זאת, ω_4, ω_5, ו-ω_6 כולם מתגלים לאיטם, משום שכל המשתנים מהביטוי הראשון חוזרים על עצמם בביטוי השני בתוספת עוד משתנה חדש, וכך הלאה.
כל רצף בעל ביטוי אחד בלבד (כמו ω_2) מתגלה לאיטו באופן ריק, ללא תלות באילו משתנים מופיעים בו.

כתבו הגדרה מונחית תחביר עבור G_2 המשתמשת בתכונות **נוצרות בלבד** שתוודא כי המילה מתארת רצף ביטויים מתגלה לאיטו.

הוראות:

- מותר להשתמש בפעולות על קבוצות כחלק מהקוד שלכם על מנת לקצר.
- אם המילה תקינה אין לבצע דבר, ואם המילה אינה תקינה יש לקרוא לפונקציה error()
- אין לשנות את הדקדוק! יש לכתוב פעולות סמנטיות רק במקומות המסומנים לכם.
- אין להוסיף לטרמינלים תכונות סמנטיות. לטרמינל id תכונה בשם $name$ המכילה את הלקסמה שלו. לטרמינל num תכונה בשם $value$ המכילה את הערך המספרי של הקבוע.

ציינו במפורש איזה תכונות סמנטיות הגדרתם לכל אחד מהמשתנים **ונמקו היטב!!** את משמעות התכונה.

מלאו את מימוש הבדיקה שלכם עבור כל כללי הדקדוק.

יש למלא את ההגדרות במקום המתאים לכך במחברת התשובות.

שאלה 1 - דקדוקים -

א) ידוע לנו שדקדוק G הוא ב-LL(1) $\Leftrightarrow$ לכל שני כללים ב-G
השייכים לאותו משתנה A מתקיים:

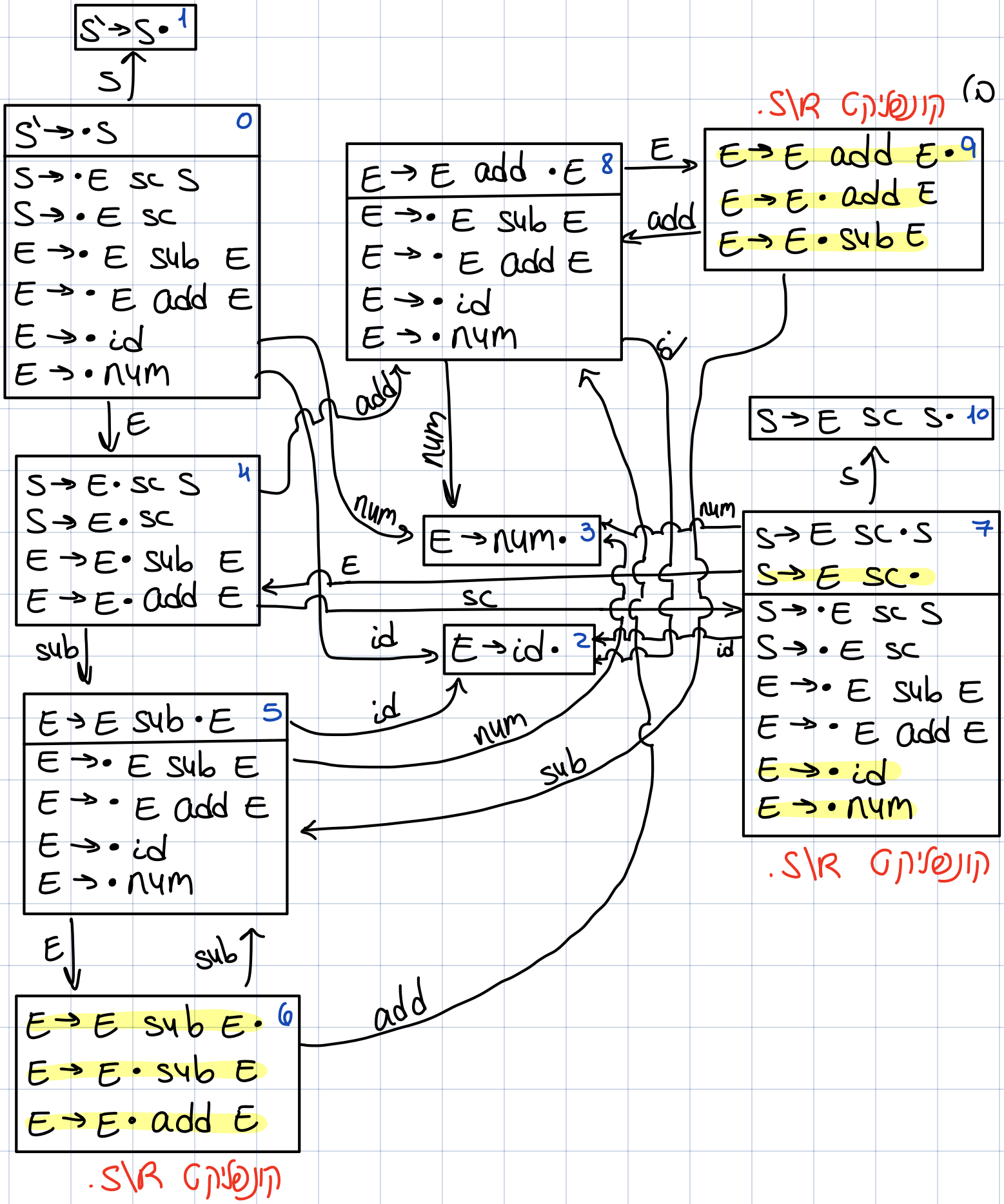
$$select(A \to \alpha) \cap select(A \to \beta) = \phi$$

לכן, נסתכל על:

select(Binop → add E E) = first(add E E) = {add}
select(Binop → sub E E) = first(sub E E) = {sub}
select(E → lpar Binop rpar) = first(lpar Binop rpar) = {lpar}
select(E → id) = first(id) = {id}
select(E → num) = first(num) = {num}

ניתן לראות שהחיתוך בין כל שתי הקבוצות של אותו משתנה
ריק ולכן לא יהיו התנגשויות ולפי המשפט הדקדוק ב-LL(1).

ג)



קונפליקט S\R.

קונפליקט S\R.

קונפליקט S\R.

ג)

```
%nonassoc sc
%left add
%right sub
%nonassoc num
%nonassoc id
```

בשביל לפתור את הקונפליקט בין $E \to E\ sub\ E \bullet$
ל- $E \to E \bullet sub\ E$ נגדיר את sub עם אסוציאטיביות שמאלית.
בשביל לפתור את הקונפליקט בין $E \to E\ add\ E \bullet$
ל- $E \to E \bullet add\ E$ נגדיר את add עם אסוציאטיביות שמאלית.
בשביל לפתור את הקונפליקט בין $E \to E\ sub\ E \bullet$
ל- $E \to E \bullet add\ E$ נגדיר את sub עם עדיפות גדולה יותר ולכן
נגדיר אותו מתחת ל- add ברשימה.
לבסוף, נרצה שלאסימונים id ו-num תהיה עדיפות גדולה יותר
משל sc כדי שנעדיף לבצע shift ולא reduce ולכן נשים
את id ו-num מתחת ל- sc ברשימה.

ד) התכונות של S: set של המשתנים המופיעים בביטוי השמאלי ביותר ב-S.

התכונות של E: set של המשתנים בביטוי.

```
S → E sc S1 {
    if (E.set ⊆ S1.set and |E.set| == |S1.set| - 1)
        S.set = E.set
    else
        error()
}

S → E sc {
    S.set = E.set
}

E → E1 sub E2 {
    E.set = E1.set ∪ E2.set
}
```

$E \rightarrow E_1 \text{ add } E_2 \{$

$\quad E.set = E_1.set \cup E_2.set$

$\}$

$E \rightarrow id \{$

$\quad E.set = \{id\}$

$\}$

$E \rightarrow num \{$

$\quad E.set = \{\}$

$\}$

שאלה 2: אנליזה סטטית (30 נק')

נוסיף לשפת הרביעיות את הפקודה call המבצעת קריאה לפונקציה. על מנת לאפשר העברת פרמטרים וערכי החזרה, נוספות גם שתי הפקודות:

`param x` דוחפת את הערך של x למחסנית כפרמטר

`retval x` קוראת את ערך החזרה של הפונקציה מהמחסנית לתוך x

נתון הקוד הבא לפונקציה `analyze_me`:

שימו לב כי x* היא פניה לזיכרון בכתובת x, כפי שראיתם בהרצאה.

```
1.  analyze_me: //params: b, r
2.      if r == 8 goto end
3.      i = 0
4.  loop:
5.      if i >= 8 goto loop_end
6.      param b
7.      param r
8.      call isSafe
9.      retval t1
10.     if t1 == 0 goto loop_post
11.     t1 = r * 4
12.     t1 = t1 + b
13.     *t1 = i //write to memory address
14.     t2 = t1 + 1
15.     param b
16.     param t2
17.     call analyze_me
18.     *t1 = -1 //write to memory address
19. loop_post:
20.     i = i + 1
21.     goto loop
22. loop_end:
23. end:
24. nop
```

א. (5 נק') כאשר תרחיבו את ייצור קוד האסמבלי משפת הרביעיות לטפל בפקודה call, איזה טיפול עליכם להוסיף עבור רגיסטרים? הסבירו:

1. היכן בקוד יידרש טיפול? הסבירו בכלליות וגם ציינו את מספרי השורות בפונקציה `analyze_me` בהן הטיפול ייתווסף.
2. מדוע הטיפול נדרש?

אחרי הרצת אלגוריתם register allocation, התקבלו ההקצאות הבאות:

משתנה	רגיסטר
b	r5
i	r8
r	r9
t1	r17
t2	r24

בתיעוד ה-ABI שהקומפיילר שלכם משתמש בו נתונה לכם החלוקה הבאה בין הרגיסטרים:

Caller-save	Callee-save
r8-r15 r24-r25	r1-r7 r16-r23

ב. (4 נק') מהו הטיפול הטוב ביותר ברגיסטרים שניתן לבצע **<u>ללא כל מידע נוסף</u>**? הסבירו בקצרה את החיסרון של הטיפול שהצעתם.

ג. (6 נק') הגדירו אנליזה שתאפשר לכם לשפר את הטיפול. ניתן להשתמש באנליזה שלמדתם בכיתה אך יהיה צורך להרחיב אותה לפקודות החדשות, או להגדיר חדשה.
1. ציינו את הדומיין של האנליזה, את יחס הסדר ופעולת ה-join, ואת פונקציית המעברים.
2. הסבירו כיצד תשתמשו בתוצאת האנליזה כדי לשפר את הטיפול.

ד. (10 נק') ציירו את ה-CFG של הפונקציה והריצו את האנליזה.

ה. (5 נק') <u>לפי תוצאות האנליזה שלכם</u>, הסבירו כיצד יטפל הקומפיילר ברגיסטרים בנקודות שהגדרתם.

שאלה 2 - אנליזה -

(א) 1) זיכוי רגיסטרים בתחילת כל שונקציה, בדוגמה בין שורה 1 ל-2.
נשתנה את הרגיסטרים מיציאה מכל שונקציה, בדוגמה אחרי שורה 23.
זיכוי רגיסטרים לפני כל קריאה לשונקציה (לפני call), בדוגמה בין שורה 8 ל-9 ובין שורה 16 ל-17.
נשתנה את הרגיסטרים בתגובה מהקריאה (אחרי call), בדוגמה בין שורה 9 ל-10 ובין שורה 17 ל-18.
2) השונקציה הנקראת עלולה להשתמש ברגיסטר ולדרוס את התוכן שלו וכבר כשנתבונן לשונקציה הקוראת היא עלולה להשתמש ברגיסטר ולהניח שהתוכן בתוכו הוא התוכן שהיא נתנה לו ולא התוכן שהשונקציה הקוראת שמה בו.

(ב) בהתחשב ברגיסטרים שמוקצים בכל שונקציה נעשה:
בתחילת השונקציה נזכה את רגיסטרים r5 ו-r17 ונשתנה אותם בסוף השונקציה.
לפני הקריאה לשונקציה נזכה את r28,r29 ו- r24 ונשתנה אותם בתגובה מהשונקציה.
החיסרון הוא שאנו מזכים רגיסטרים שלא בטוח שיש בהם צורך.

③ 1) נשתמש באוניברסלית חיות אשר נמצאה בכיתה.

הדומיין, יחס הסדר וה-join לא משתנים.

פונקציית המעברים בהה לנמצא בכיתה עם הרחבה

לפקודות החדשות:

	gen	kill
param x	{x}	ϕ
retval x	ϕ	{x}
call func	ϕ	ϕ
*x = y	{x,y}	ϕ

2) כאשר ניגש לפקודת call אותנו נסתכל על ה-out או על ה-in (כי פעולת call לא משפיעה על האוניברסלית) של הגרף ואותנו נצבע רק את הרגיסטרים שנמצאים ב-out של הגרף (כלומר, רק לרגיסטרים שחיים לאחר הקריאה) והם caller saved.

אם הרגיסטרים לא חיים לאחר הקריאה הרי שאין צורך לצבוע את הרגיסטרים האלו כיוון ולא נעשה בהם שימוש ולכן ניתן לדרוס את הערך בו.

③

	out	in
if r == 8 goto end	{b,r}	{b,r}
i=0	{b,r}	{i,b,r}
if i >= 8 goto loop_end	{i,b,r}	{i,b,r}
param b	{i,b,r}	{i,b,r}
param r	{i,b,r}	{i,b,r}
call issafe	{i,b,r}	{i,b,r}
retval t1	{i,b,r}	{i,b,r}
if t1==0 goto loop_post	{i,b,r}	{i,b,r}
t1= r*4	~~{i,b,r}~~	~~{i,t1,b}~~
	{i,b,r}	{i,b,r,t1}
t1 = t1+b	~~{i,t1,b}~~	~~{i,t1,b}~~
	{i,b,r,t1}	{i,b,r,t1}
*t1 = i	~~{i,t1,b}~~	~~{i,t1,b}~~
	{i,b,r,t1}	{i,b,r,t1}
t2 = t1+1	~~{i,t1,b}~~	~~{i,t1,t2,b}~~
	{i,b,r,t1}	{i,b,r,t1,t2}
param b	~~{i,t1,t2,b}~~	~~{i,t1,t2}~~
	{i,b,r,t1,t2}	{i,b,r,t1,t2}
param t2	~~{i,t1,t2}~~	~~{i,t1}~~
	{i,b,r,t1,t2}	{i,b,r,t1}
call analyze_me	~~{i,t1}~~	~~{i,t1}~~
	{i,b,r,t1}	{i,b,r,t1}
*t1 = -1	~~{i,t1}~~	~~{i}~~
	{i,b,r,t1}	{i,b,r}
i= i+1	~~{i}~~	~~ϕ~~
	{i,b,r}	{i,b,r}
goto loop	~~ϕ~~	~~ϕ~~
	{i,b,r}	{i,b,r}
nop	ϕ	ϕ

ה) לפי תוצאות האופטימיזציה לפני הקריאה ל- isSafe רק i,b,r
חיים (נמצאים ב- out של הבלוק) ומתוכם רק i ו- r
הם שם caller save ולכן נצטרך רק את הרגיסטרים שלהם
ונשתמר אותם לאחר הקריאה.
בנוסף, לפני הקריאה ל- analyze_me רק i,b,r,t1
חיים (נמצאים ב- out של הבלוק) ומתוכם רק i ו- r הם
שם caller save ולכן נצטרך רק את הרגיסטרים שלהם ונשתמר
אותם לאחר הקריאה.
סה"כ אנחנו רואים שאין צורך לשמור את הרגיסטר של t2.

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

<u>הגדרה:</u> דקדוק G הוא LL(1) <u>אם ורק אם</u> לכל שני כללים ב- G השייכים לאותו משתנה A מתקיים:
$\text{select}(A \to \alpha) \cap \text{select}(A \to \beta) = \varnothing$

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

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:

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

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:

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

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 מהצורה $G = (V, E)$:

הצורה הכללית של המשוואות בחישוב סריקה קדמית:

$$\mathrm{in}(B) = \bigsqcup_{(S,B)\in E} \mathrm{out}(S)$$
$$\mathrm{out}(B) = f_B\big(\mathrm{in}(B)\big)$$

הצורה הכללית של המשוואות בחישוב סריקה אחורית:

$$\mathrm{in}(B) = \bigsqcup_{(B,D)\in E} \mathrm{out}(D)$$
$$\mathrm{out}(B) = f_B\big(\mathrm{in}(B)\big)$$

שפת FanC

אסימונים:

תבנית	אסימון
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
;	SC
(	LPAREN
)	RPAREN
{	LBRACE
}	RBRACE
=	ASSIGN
== | != | < | > | <= | >=	RELOP
+ | - | * | /	BINOP
[a-zA-Z][a-zA-Z0-9]*	ID
0 | [1-9][0-9]*	NUM
"([^\n\r\"\\]|\\[rnt"\\])+"	STRING

דקדוק:

1. $Program \rightarrow Statements$
2. $Statements \rightarrow Statement$
3. $Statements \rightarrow Statements\ Statement$
4. $Statement \rightarrow LBRACE\ Statements\ RBRACE$
5. $Statement \rightarrow Type\ ID\ SC$
6. $Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
7. $Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
8. $Statement \rightarrow Call\ SC$
9. $Statement \rightarrow RETURN\ SC$
10. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
11. $Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
12. $Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
13. $Statement \rightarrow BREAK\ SC$
14. $Statement \rightarrow CONTINUE\ SC$
15. $Call \rightarrow ID\ LPAREN\ Exp\ RPAREN$
16. $Type \rightarrow INT$
17. $Type \rightarrow BYTE$
18. $Type \rightarrow BOOL$
19. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
20. $Exp \rightarrow Exp\ BINOP\ Exp$
21. $Exp \rightarrow ID$
22. $Exp \rightarrow Call$
23. $Exp \rightarrow NUM$
24. $Exp \rightarrow NUM\ B$
25. $Exp \rightarrow STRING$
26. $Exp \rightarrow TRUE$
27. $Exp \rightarrow FALSE$
28. $Exp \rightarrow NOT\ Exp$
29. $Exp \rightarrow Exp\ AND\ Exp$
30. $Exp \rightarrow Exp\ OR\ Exp$
31. $Exp \rightarrow Exp\ RELOP\ Exp$
32. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$