מבחן סוף סמסטר – מועד א'

טור מקור

מרצה אחראי: ד"ר שחר יצחקי

מתרגלים: הילה לוי, תומר כהן, גיא ארבל, אנדריי בבין

הוראות:

1. בטופס המבחן 13 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 התשובות הנפרד בלבד. את התשובות לשאלות הפתוחות יש לכתוב במקומות המסומנים בטופס המיועד לכך.
8. במידה ואתם משתמשים בדפי העזר הנוספים יש לציין במפורש את מספר השאלה ומספר הסעיף.
9. ודאו כי אתם מגישים טפסי התשובות המיועדים בלבד (את מחברת הטיוטה ניתן לשמור אצלכם).

בקובץ זה בחלק האמריקאי התשובה הנכונה בכל השאלות היא תשובה א'.

שימו לב, שינויים שנאמרו בעל פה בזמן המבחן מסומנים בצהוב.

בהצלחה!

חלק א' - שאלות סגורות (45 נק')

שלבי קומפילציה (27 נק')

נתונה התוכנית הבאה בשפת FanC:

```
1.  int m = 100;
2.  int n = 200;
3.  int x = n * m;
4.  int y = (n + m) / 2;
5.  int k = 0;
6.  while (k * k < x) {
7.      k = k + 1;
8.  }
9.  if (k < y){
10.     print("true");
11. }
12. else {
13.     print("false");
14. }
```

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 1 ב-

```
intm = 100;
```

א. שגיאה בניתוח הסמנטי

ב. שגיאה בניתוח תחבירי

ג. שגיאה בניתוח לקסיקלי

ד. אין שגיאה

ה. שגיאה בייצור קוד

ו. שגיאה בזמן ריצה

שאלה מספר 2:

(3 נק') מחליפים את שורה 5 ב-

```
int k = "0";
```

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 4 ב-

```
int y = {n + m} / 2;
```

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 9 ב-

```
if (k - y) {
```

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

שאלה מספר 5:
(3 נק') מחליפים את שורה 10 ב-

```
print("true);
```

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

שינויים ב-FanC- שאלות 6-7:

נרצה להוסיף לשפת FanC תמיכה בפונקציות אשר מקבלות ארגומנט **בודד** ומחזירות ערך (בודד). הגדרת הפונקציות דומה לשפת C, אך ההגדרות משולבות בין הצהרות ומשפטים אחרים בתוכנית, למשל:

```
1. printi(10);
2. int foo(int x) {
3.         return x + 2;
4. }
5. printi(foo(3));
6. int bar(int y) {
7.             return foo(y);
8. }
9. printi(bar(foo(4)));
```

הערה:
טיפוס הארגומנט וטיפוס ערך החזרה הם int, byte או bool.

שאלה מספר 6:

(6 נק') בשביל תמיכה בפיצ'ר החדש, באילו שלבי קומפילציה מבין הרשומים כאן, **לא** נבצע שינויים?

א. ניתוח לקסיקלי

ב. ייצור קוד

ג. ניתוח סמנטי

ד. נצטרך לערוך שינויים בכל אחד משלבי הקומפילציה שרשומים כאן.

שאלה מספר 7:

(6 נק') נתבונן בקוד הבא:

```
1. int a = 2 / 0;
2. int f(int x) {
3.      int y = x * 2;
4.      return y * y;
5. }
6. int y = f(2);
```

האם יווצר קוד של פונקציה f בשפת היעד כאשר לא משתמשים באופטימיזציות?

א. כן.

ב. לא, הקוד לא יתקמפל בגלל הגדרה של משתנה y בשורה 6 אשר כבר הוגדר בשורה 3.

ג. לא, בזמן ריצה תתבצע חלוקה ב-0 בשורה 1 ולכן התוכנית תקרס לפני שתגיע להגדרה של f.

ד. לא, הקוד הנתון לא יתקמפל כי חסרה פונקציית main.

אופטימיזציות (18 נק')

נתונה התוכנית הבאה בשפת הרביעיות שבה נמחקה כתובת הקפיצה משורה 5.

```
1.  a = 0
2.  b = 1000
3.  c = 0
4.  if c >= b goto 10
5.  if d == 0 goto _____
6.  if e == 0 goto 8
7.  a = c + a
8.  c = c + 1
9.  goto 4
10. print a
```

שאלה מספר 8:
(6 נק') עבור הערך 2 בשורה 5 מה מספר הבלוקים הבסיסיים?

א. 8

ב. 6

ג. 7

ד. אף תשובה אינה נכונה

שאלה מספר 9:
(6 נק') מבין כל הערכים האפשריים עבור כתובת בשורה 5, מהו **מספר** הבלוקים הבסיסיים הקטן ביותר שניתן לקבל?

א. 7

ב. 6

ג. 8

ד. אף תשובה אינה נכונה

שאלה מספר 10:

(6 נק') מהנדס בכיר רצה לבדוק את זמני הריצה של הקוד GreatC (שפה דמוית C) שלו. נתון הקוד הבא, שימו לב כי הערכים בשורה של הגדרת המערך x נמחקו.

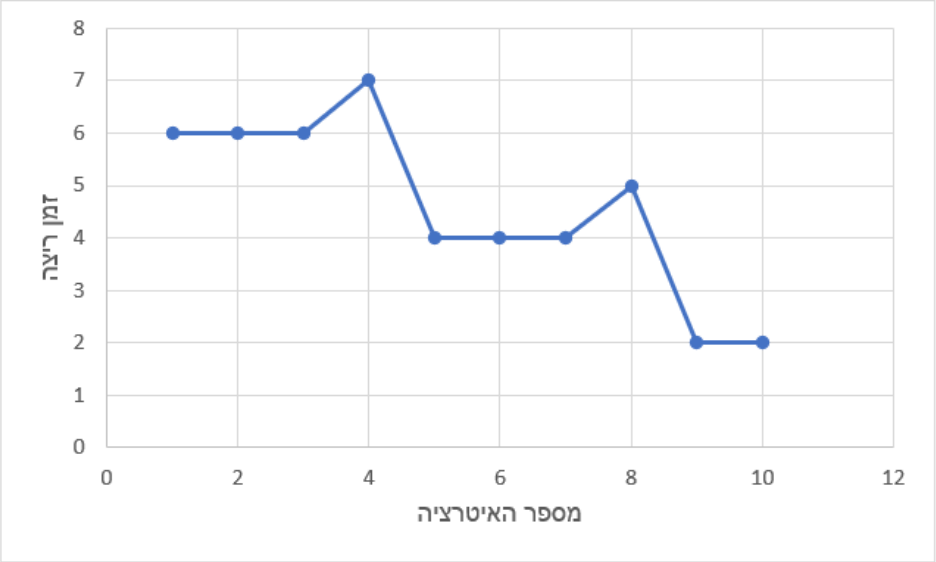
```
int main() {
        int x[10] = {_,_,_,_,_,_,_,_,_,_};
        int i = 0;
        while(i < 10) {
              switch(a[i] x[i]) {
                    case 1:
                          f();
                          break;
                    case 2:
                          g();
                          break;
                    default:
                          h();

              }
        i++;
        }
        Return 0;
}
```

הקוד קומפל ל-LLVM IR באמצעות הקומפיילר של GreatC וכמו כן הורץ באמצעות אינטרפרטר של LLVM IR עם כל האופטימיזציות שנלמדו בקורס, כולל JIT.

ידוע כי f, g, h **אינן** קוראות אחת לשנייה **ואינן** קוראות לפונקציות משותפות.

בנוסף, ידוע כי זמן הריצה של f, g, h **זהה** בכל אחת מרמות האופטימיזציה שלהן.

נתון גרף הביצועים הבא כאשר ציר ה x (הציר ~~האנכי~~ האופקי) מייצג את מספר האיטרציה של לולאת ה while, וציר ה y (הציר ~~האופקי~~ האנכי) הוא זמן הריצה של אותה איטרציה.



מבין המערכים הבאים מה המערך שהכי סביר שנמחק מן השורה הראשונה (מספור המערך משמאל לימין)?

א. 1,1,1,1,1,1,1,1,1,1
ב. 1,1,1,2,1,1,1,3,1,1
ג. 1,1,1,2,1,1,1,2,1,1
ד. 1,1,1,2,2,2,2,3,3,3

חלק ב' - שאלות פתוחות (55 נק')

שאלה 1: דקדוקים (25 נק')

הדקדוק הבא מייצג את שפת הפקודות שניתנות לצ'ארלי כאשר המשתנה ההתחלתי הוא WUFF.
שימו לב, משתנים מיוצגים באותיות גדולות וטרמינלים עם קו תחתון.

$$WUFF \rightarrow CMD$$
$$CMD \rightarrow CMD\ \underline{down}$$
$$CMD \rightarrow \underline{sit}\ CMD$$
$$CMD \rightarrow \underline{good!}$$

א. (10 נק') צ'ארלי החכמה מנסה להבין את השפה, אך לא מצליחה כי הדקדוק לא שייך למנתח LR(1).
עזרו לצ'ארלי להבין למה הדקדוק לא שייך ל-LR(1).

ב. (10 נק') עזרו לצ'ארלי לשנות את הדקדוק מבלי לשנות את השפה כך שיתאים לאחד מבין המנתחים הבאים: LR(0), SLR, LR(1). יש להוכיח כי המנתח אכן מקבל את הדקדוק שיצרתם על ידי בניית אוטומט וטבלת מצבים.
ענו על הסעיף כך שהמנתח יצרוך כמות נמוכה ככל הניתן של זיכרון, נגדיר כמות זיכרון שהמנתח צורך בתור מספר התאים בטבלת הניתוח.

ג. (5 נק') צ'ארלי קיבלה את הקלט הבא:

$$\underline{sit}\ \underline{good!}\ \underline{down}$$

עזרו לצ'ארלי לנתח את המילה, הציגו את כל מצבי המחסנית במהלך הניתוח.

פתרון שאלה 1: דקדוקים

א. נצייר רק את המסלול המוביל לקונפליקט:

$$\begin{array}{c} S' \to \cdot\, WUFF\,,\$ \\ WUFF \to \cdot\, CMD\,,\$ \\ CMD \to \cdot\, CMD\ down\,,\$ \\ CMD \to \cdot\, sit\ CMD\,,\$ \\ CMD \to \cdot\, good!\,,\$ \\ CMD \to \cdot\ CMD\ down\,,down \\ CMD \to \cdot\, sit\ CMD\,,down \\ CMD \to \cdot\, good!\,,down \end{array}$$

$\xrightarrow{sit}$

$$\begin{array}{c} CMD \to sit \cdot CMD\,,\$ \\ CMD \to sit \cdot CMD\,,down \\ CMD \to \cdot\, CMD\ down\,,\$ \\ CMD \to \cdot\, sit\ CMD\,,\$ \\ CMD \to \cdot\, good!\,,\$ \\ CMD \to \cdot\ CMD\ down\,,down \\ CMD \to \cdot\, sit\ CMD\,,down \\ CMD \to \cdot\, good!\,,down \end{array}$$

$\xrightarrow{CMD}$

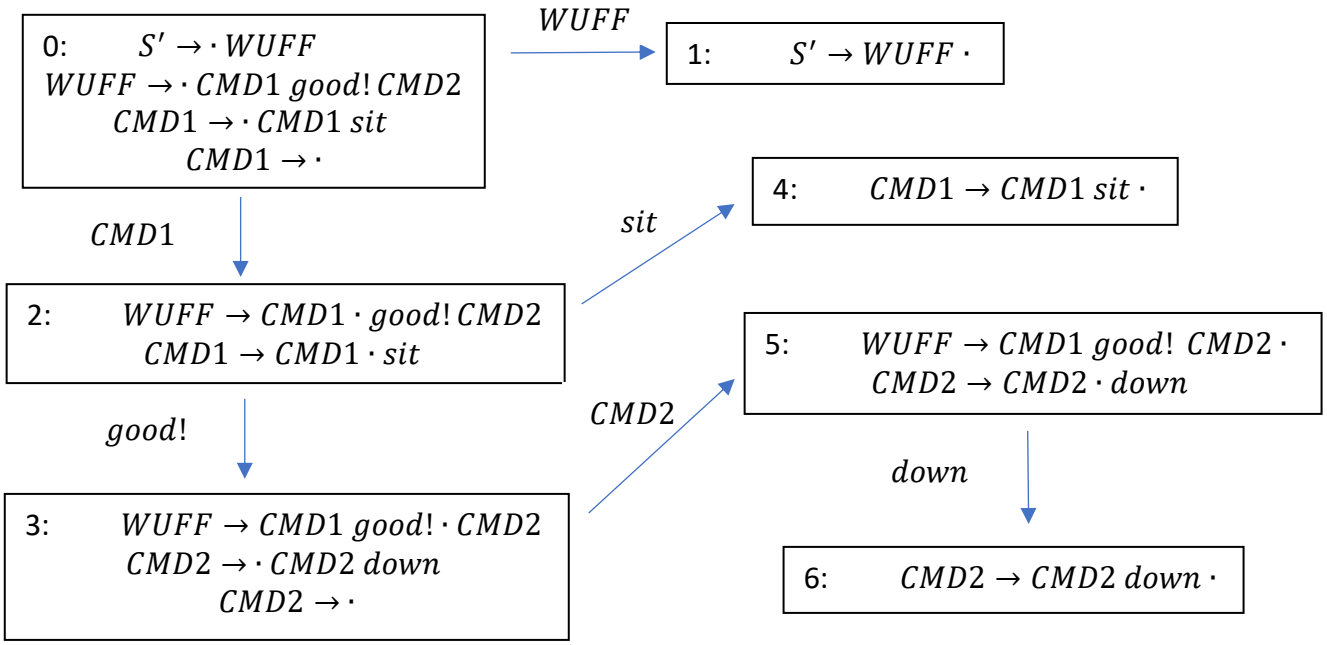
$$\begin{array}{c} CMD \to sit\ CMD \cdot\,,\$ \\ CMD \to sit\ CMD \cdot\,,down \\ CMD \to CMD \cdot down\,,\$ \\ CMD \to CMD \cdot\ down\,,down \end{array}$$

ניתן לראות כי המצב מטה יש קונפליקט shift/reduce עבור הטרמינל down.

ב. ישנן כמה דרכים אפשריות ליצור דקדוק בסעיף ב', דקדוק לדוגמה הוא:

1: $WUFF \to CMD1\ good!\ CMD2$
2: $CMD1 \to CMD1\ sit$
3: $CMD1 \to \epsilon$
4: $CMD2 \to CMD2\ down$
5: $CMD2 \to \epsilon$

דקדוק זה שייך לSLR.
נבנה אוטומט:



נשים לב לfollow של כל משתנה:

$$follow(WUFF) = \{\$\}$$
$$follow(CMD1) = \{good!, sit\}$$
$$follow(CMD2) = \{\$, down\}$$

נבנה את טבלת המצבים:

State	Action				GOTO		
	sit	good!	down	$	WUFF	CMD1	CMD2
0	r3	r3			1	2	
1				acc			
2	s4	s3					
3			r5	r5			5
4	r2	r2					
5			s6	r1			
6			r4	r4			

נשים לב שאין קונפליקטים ולכן הדקדוק שיצרנו אכן שייך למנתח SLR.

ג. ננתח את המילה לפי הדקדוק שיצרנו בסעיף ב׳:

Step	stack	input	action
1	(0,)	sit good! down	r3
2	(0,) (CMD1, 2)	sit good! down	s4
3	(0,) (CMD1, 2) (sit, 4)	good! down	r2
4	(0,) (CMD1, 2)	good! down	s3
5	(0,) (CMD1, 2) (good!, 3)	down	r5
6	(0,) (CMD1, 2) (good!, 3) (CMD2, 5)	down	s6
7	(0,) (CMD1, 2) (good!, 3) (CMD2, 5) (down, 6)		r4
8	(0,) (CMD1, 2) (good!, 3) (CMD2, 5)		r1
9	(0,) (WUFF, 1)		acc

שאלה 2: אנליזה סטטית (30 נק')

אלפור משתמש במודול ייצור קוד ביניים אשר הפלט שלו הוא קוד בשפת הרביעיות (3AC). אלפור מעוניין לבצע אופטימיזציות ולאחר מכן לתרגם את קוד הביניים לאסמבלי (של מעבד אלפא, כמובן). אך אבוי – הנוהל המקובל קובע כי קוד הביניים צריך להיות בצורת SSA, ואילו הפלט של המודול שבידי אלפור איננו בצורה זו. למשל, הוא קיבל ממנו את שלוש התוכניות הבאות, שבהן למשתנים w, m, i ו-sum יש יותר מהשמה אחת.

abs

```
1:  w := readi()
    if w >= 0 goto 3
2:  w := -w
3:  printi(w)
```

max

```
1:  a := readi()
    b := readi()
    goto 2
2:  if a < b then
    goto 4
    if a < b goto 4
3:  m := a - b
    goto 5
4:  m := b - a
5:  printi(m)
```

sum_squares

```
1:  n := readi()
    i := 1
    sum := 0
2:  if i >= n goto 5
3:  t := i * i
    sum := sum + t
4:  i := i + 1
    goto 2
5:  printi(sum)
```

אלפור קרא בספר כי ניתן להמיר כל קוד ביניים הכולל השמות לקוד SSA על-ידי שימוש באנליזה סטטית מסוג Reaching Definitions. האנליזה היא מסוג gen/kill המוגדרת באופן הבא:

Statement at ℓ	kill(ℓ)	gen(ℓ)
v := expr	{ (v,i) | i ∈ Lab }	{ (v,ℓ) }
anything else	∅	∅

א. (5 נק') שרטטו את גרף בקרת הזרימה (CFG) של כל אחת משלוש התוכניות שלמעלה (לפי החלוקה לבלוקים שנתונה בשאלה).

ב. (5 נק') הריצו את האנליזה של Reaching Definitions על הגרפים, ומלאו את התוצאות הסופיות בטבלה (in ו-out של כל בלוק בסיסי).

ג. (10 נק') בצעו את ההליך המתואר בפרוצדורה מהספר של אלפור על הקוד של שתי התוכניות השמאליות מהדוגמה (abs, max).

הפרוצדורה בספר של אלפור קובעת:

(1) מותר למהדר להניח כי בכל בלוק בסיסי לא תהיינה שתי השמות שונות לאותו משתנה.

(2) החליפו כל הגדרה (definition) של משתנה בהגדרה של משתנה חדש, על פי הקונבנציה הבאה: השמה למשתנה v בבלוק ℓ תהפוך להשמה למשתנה בשם v_ℓ

(3) החליפו כל שימוש (use) במשתנה v בשימוש במשתנה v_ℓ על-פי התוצאה של Reaching Definitions; מוצאים את ההגדרה של v שמגיעה לאותו שימוש, (v, ℓ) (שייך ל-in(b) כאשר b הוא הבלוק שבו מופיע השימוש), וכך יודעים מיהו v_ℓ המתאים.

(4) במקרה שיש יותר מהגדרה אחת שמגיעה לשימוש כלשהו, מוסיפים הגדרה של משתנה זמני נוסף t_v_ℓ (הרישא _t היא קבועה ומציינת שמדובר בהגדרה שיוצרה על ידי הקומפיילר) בעזרת אופרטור φ (פִּי) אשר בוחר את המשתנה המתאים. לדוגמה

t_x_4 = φ(x_1, x_3)

ד. (10 נק') טוֹנטוֹ מעוניין להשתמש בתשתית LLVM לצורך תרגום לקוד אלפא. ב LLVM, כאשר משתמשים באופרטור φ, לא די לציין את שמות המשתנים המופיעים כארגומנטים, אלא יש לציין גם את הבלוק שממנו מגיעה ההגדרה של כל אחד מהארגומנטים. יתר על כן, כל ציון בלוק המוצמד לארגומנט חייב להיות בלוק קודם ישיר (direct predecessor) של הבלוק שבו מופיע השימוש באופרטור φ — כלומר חייבת להיות קשת אחת בדיוק המקשרת ביניהם, ואסורים מסלולים באורך גדול מ-1.

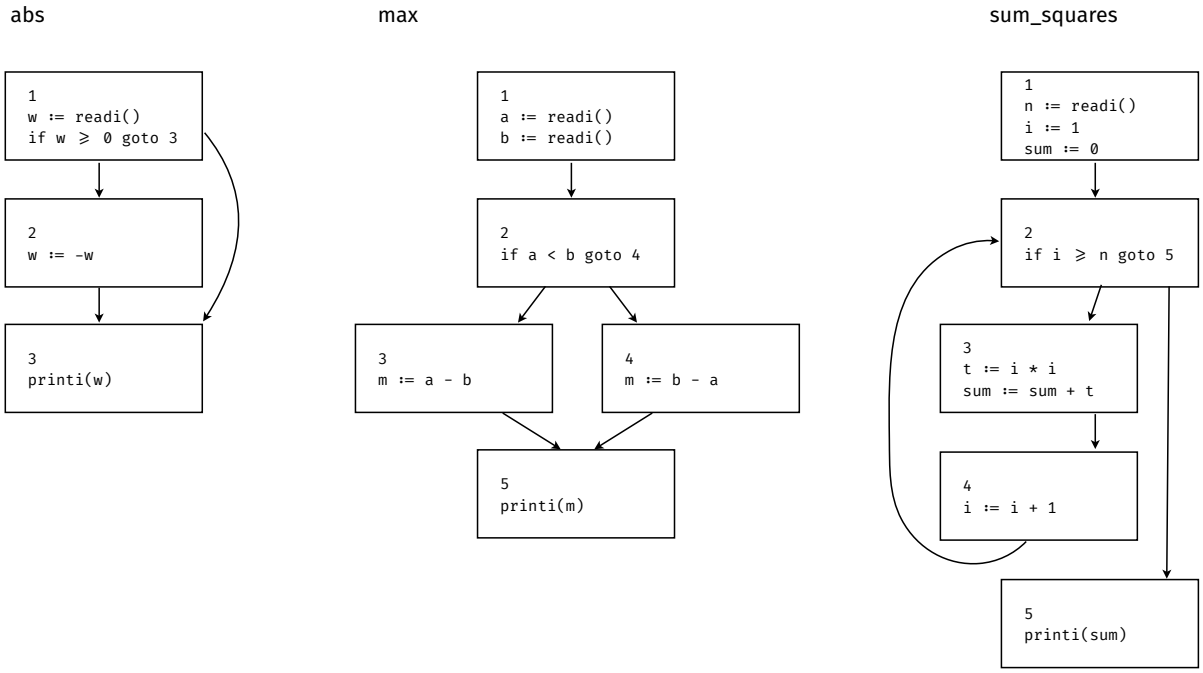
טונטו מציע להחליף כל ארגומנט v_ℓ בזוג [v_ℓ, ℓ] ובכך לספק את הדרישה, למשל

t_x_4 = φ([x_1, 1], [x_3, 3])

הציגו לטונטו תוכנית דוגמה, ואת התוצאה של הגישה המוצעת, שבה התוצאה איננה מקיימת את התנאי הנדרש שלעיל. הסבירו כיצד התנאי מופר.

פתרון שאלה 2: אנליזה סטטית

א. גרפי בקרת הזרימה של הפונקציות הנתונות



ב. תוצאות הרצת האנליזה

abs

	in	out
1	{}	(w,1)
2	(w,1)	(w,2)
3	(w,1) (w,2)	(w,1) (w,2)

max

	in	out
1	{}	(a,1), (b,1)
2	(a,1), (b,1)	(a,1), (b,1)
3	(a,1), (b,1)	(a,1), (b,1) (m,3)
4	(a,1), (b,1)	(a,1), (b,1) (m,4)
5	(a,1), (b,1) (m,3) (m,4)	(a,1), (b,1) (m,3) (m,4)

sum_squares

	in	out
1	{}	(n,1), (i,1), (sum,1)
2	(n,1), (i,1), (sum,1), (i,4), (sum,3) (t,3)	(n,1), (i,1), (sum,1), (i,4), (sum,3) (t,3)

	in	out
3	(n,1), (i,1), (sum,1), (i,4), (sum,3) (t,3)	(n,1), (i,1), (i,4), (sum,3), (t,3)
4	(n,1), (i,1), (i,4), (sum,3), (t,3)	(n,1), (i,4), (sum,3), (t,3)
5	(n,1), (i,1), (sum,1), (i,4), (sum,3) (t,3)	(n,1), (i,1), (sum,1), (i,4), (sum,3) (t,3)

ג. התוצאה של הרצת ההליך מהספר של אלפור לקבלת קוד SSA

```
1:  w_1 := readi()
    if w_1 ≥ 0 goto 3
2:  w_2 := -w_1
3:  t_w_3 := phi(w_1, w_2)
    printi(t_w_3)
```

```
1:  a_1 := readi()
    b_1 := readi()
    goto 2
2:  if a_1 < b_1 goto 4
3:  m_3 := a_1 - b_1
    goto 5
4:  m_4 := b_1 - a_1
5:  t_m_5 := phi(m_3,m_4)
    printi(t_m_5)
```

ד. אפשר להדגים את הבעיה בעזרת התוכנית השלישית (sum_squares). לפי תוצאת האנליזה ובהתאם להצעה של טונטו מתקבל קוד הביניים הבא:

```
1:  n_1 := readi()
    i_1 := 1
    sum_1 := 0
2:  t_i_2 := φ([i_1,1], [i_4,4])
    if t_i_2 ≥ n goto 5
3:  t_i_3 := φ([i_1,1], [i_4,4])
    t_sum_3 := φ([sum_1,1], [sum_3,3])
    t_3 := t_i_3 * t_i_3
    sum_3 := t_sum_3 + t_3
4:  t_i_4 := φ([i_1,1], [i_4,4])
    i_4 := t_i_4 + 1
    goto 2
5:  t_sum_5 := φ([sum_1,1], [sum_3,3])
    printi(t_sum_5)
```

ההפעלות של **φ** המופיעות באדום מפרות את התנאי של LLVM IR מכיוון שמספרי הבלוקים המופיעים אינם בלוקים קודמים ישירים (למשל, אין קשת יחידה בין 1 לבין 3).
(ההפעלה בבלוק 2 היא תקינה כי גם 1 וגם 4 הם קודמים ישירים של 2)

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

<u>הגדרה:</u> דקדוק G הוא LL(1) <u>אם ורק אם</u> 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 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. $Exp \rightarrow LPAREN\ Type\ RPAREN\ Exp$