שאלה 2 (30 נקודות): אנליזה סטטית

האנליזה Available Expressions (שנלמדה בכיתה) נתונה על ידי פונקצית המעברים הבאה:

Statement	out(ℓ)
x := a	in(ℓ) \ {a' ∈ AExp | x ∈ FV(a')} ∪ {a' ∈ AExp(a) | x ∉ FV(a')}
cond	in(ℓ) ∪ AExp(*cond*)

כאשר:

AExp = קבוצת כל הביטויים האריתמטיים **שמופיעים בתכנית**.

AExp(e) = קבוצת כל הביטויים האריתמטיים המופיעים בביטוי e (כולל e עצמו, אם e הוא ביטוי אריתמטי).

FV(e) = קבוצת המשתנים המופיעים בביטוי e.

in(ℓ) ו-out(ℓ) הם קבוצות של ביטויים אריתמטיים. הפעולה ⊔ (join) מוגדרת כ-∩ (חיתוך קבוצות).

קרטמן מציע לשפר את האנליזה על ידי טיפול במקרה מיוחד:

Statement	out(ℓ)
x := x + n	{a[x – n / x] | a ∈ in(ℓ)} when n is a constant

הפעולה a[e/x] מתארת החלפה של כל המופעים של x שנמצאים ב-a על-ידי הביטוי e. למשל:

(2 * x * y)[x – 3/x] = 2 * (x – 3) * y

שאר המקרים נשארים ללא שינוי.

א. (5 נק') האם האנליזה הזו היא מסוג Gen/Kill? הסבירו.

האנליזה עבור המקרה x := x + n, כפי שנכתבה, אינה בצורה של Gen/Kill. למרות שאפשר להמיר את הביטוי לכזה שבנוי על חיסור ואיחוד קבוצות, נסיון לעשות זאת יוביל לביטוי מהצורה:

in(ℓ) \ {a | a ∈ in(ℓ), x ∈ FV(a)} ∪ {a[x – c / x] | a ∈ in(ℓ)}

כפי שלמדנו, אסור לקבוצות ה-Gen וה-Kill להיות תלויות במצב הקלט in(ℓ). לכן זאת לא הצגה תקינה.

ב. (10 נק') קייל טוען שאנליזת קרטמן שהוצגה **אינה נאותה**. הסבירו מדוע, והדגימו על התכנית הבאה:

1: a := x * y
2: x := x + 3
3: x := z

אנליזה לא נאותה היא כזאת שמייצרת טענות **שאינן נכונות** עבור כל הריצות. במקרה של אנליזת available expressions, האנליזה של קרטמן לא נאותה בגלל שהיא טוענת שביטויים מסויימים הם זמינים, בעוד שהם אינם. (יש לזכור שביטוי נחשב זמין אם מהפעם האחרונה שהביטוי חושב, לא השתנה ערכו בעקבות השמה לאחד מהמשתנים שבו).

בתכנית לדוגמה הנתונה:

in(1) = { }	a := x * y	out(1) = {x * y}
in(2) = {x * y}	x := x + 3	out(2) = {(x – 3) * y}
in(3) = {(x – 3) * y}	x := z	out(3) = {(x – 3) * y}

במקרה של שורה 3, הביטוי (x – 3) * y עדיין מופיע בקבוצת ה-out, מכיוון ש (x – 3) * y ∉ AExp, ולכן הוא אינו מושפע מרכיב ה-Kill של פונקצית המעברים המתאימה להשמה.

זאת למרות, שערכו של x השתנה ולפיכך הערך האחרון שחושב (בשורה 1) אינו עדכני.

ג. (5 נק') עזרו לקרטמן לתקן את האנליזה שלו כך שתהיה נאותה.

כדי לתקן את האנליזה כך שתהיה נאותה (ועדיין יישמר הטיפול במקרה המיוחד) יש לדאוג למחיקת כל הביטויים שכוללים משתנה מסוים, כאשר מתבצעת השמה למשתנה, כולל כאלה שהתווספו כתוצר לוואי של החלפה בביטוי קיים. שתיים מן הדרכים לעשות זאת הן:

1. לשנות את ההגדרה של AExp כך שתכלול את כל הביטויים האריתמטיים (קבוצה אינסופית)
2. לשנות את הטיפול במשפט x := a, על ידי החלפת AExp ב-in(ℓ) בהגדרה של קבוצת ה-Kill.

ד. (10 נק') נתונה הלולאה הבאה (בצורת 3-address code):

```
1:  i := 0
2:  t1 := 4 * i
3:  if i >= n goto 10
4:  t2 := arr + t1
5:  t3 := *t2
6:  if t3 < 0 goto 10
7:  i := i + 1
8:  t1 := 4 * i
9:  goto 3
10: param t3
11: call print
```

הראו כיצד ניתן, בעזרת אנליזת קרטמן (המתוקנת), להחיל על התכנית strength reduction – כלומר להחליף פעולה איטית בפעולה מהירה יותר.

הניחו שהקומפיילר מסוגל להפעיל פישוטים אלגבריים בסיסיים על ביטויים אריתמטיים.

בשורה 8 בתכנית מתבצעת פעולת כפל 4 * i. בעזרת האנליזה של קרטמן נקבל שהביטוי 4 * (i – 1) ∈ in(8) זמין. זאת מכיוון ש-4 * i ∈ in(7), ובשורה 7 מבצעים החלפה של i ב-(i – 1). על ידי פישוט אלגברי נקבל שהביטוי הזמין שווה ל-4 * i – 4, לפיכך נוכל להחליף את החישוב 4 * i בפעולת חיבור, t1 + 4.
(שתוצאת החישוב הקודם של הביטוי נמצאת במשתנה t1 ניתן להסיק מאנליזת reaching definitions, אבל לא נדרשתם לציין זאת בתשובה. להוסיף משתנה עזר ששומר את הערך של 4 * i או לשמור את התוצאה ברגיסטר, גם הן תשובות קבילות.)

שאלה 5 (25 נקודות): Code Generation

חלק א (15 נקודות) - Register Allocation

נתון המתודה הבאה:

```
1.  void foo() {
2.      a := b + 1
3.      d := 5
4.      if a < 10 goto 6
5.      c := d * 3
6.      goto 8
7.      c := 1
8.      e := a + b
9.      f := c * d
10.     print f
11. }
```

המתודה foo משתמשת ב6 משתנים a,b,c,d,e,f כך שהמשתנה b משמש כארגומנט למתודה.

א. (5 נק') הניחו כי נתון מעבד בעל 3 רגיסטרים (r0,r1,r2) ושכל המשתנים חיים בסוף המתודה. האם ניתן להקצות רגיסטרים למשתנים בתכנית מבלי שנצטרך לכתוב אף משתנה לזיכרון? אם כן, ציינו איזה משתנה ישמר באיזה רגיסטר. אם לא, ציינו מה מספר הרגיסטרים המינימלי שיידרש.

נשים לב שאם משתנה x כלשהו חי, אזי יש לשמר את ערכו (נדגיש ששימור ערכו של משתנה אינו שקול ליכולת לחשב את ערכו מחדש, עלינו לשמר את הערך עצמו כך שלא נצטרך לחשבו בשנית). מאחר ותנאי השאלה לא מאפשרים לנו לשמור את המשתנה בזכרון, נאלץ לשמור אותו ברגיסטר (המחסנית היא חלק מהזכרון ולכן גם בה לא נוכל להשתמש).
מאחר וכל המשתנים חיים בסעיף זה, אנו יודעים שאנחנו צריכים לשמר את ערכי כל המשתנים, ולכן בהכרח נצטרך רגיסטרים כמספר המשתנים, כלומר 6.
<u>הערה:</u> גם המשתנה b צריך רגיסטר. מאחר ואנו לא יודעים לאיזה פלטפורמה יקומפל הקוד, איננו יכולים להניח שb שמור לנו במחסנית. ראינו בתרגול דוגמה למעבד שמעביר ארגומנטים דרך רגיסטרים. כמו כן, גם אם b מועבר דרך המחסנית, לא נבצע חישובים ישירות מתוך המחסנית ונצטרך לקרוא את ערכו לרגיסטר לצורך העבודה איתו.

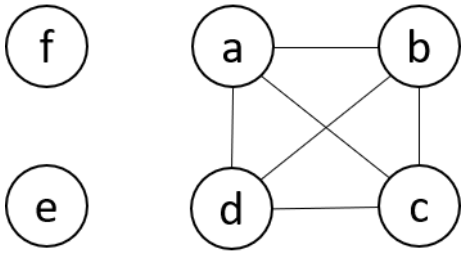
ב. (5 נק') הניחו כי נתון מעבד בעל 3 רגיסטרים (r0,r1,r2) ושאף המשתנה אינו חי בסוף המתודה, האם ניתן להקצות רגיסטרים למשתנים בתכנית מבלי שנצטרך לכתוב אף משתנה לזיכרון? אם כן, ציינו איזה משתנה ישמר באיזה רגיסטר. אם לא, ציינו מה מספר הרגיסטרים המינימלי שיידרש

בסעיף זה, מאחר ואף משתנה אינו חי, אנו יודעים שנוכל לשחרר את חלק מהרגיסטרים בהם נשתמש לצורך המשתנים ולמחזר אותם לטובת משתנים אחרים. בשאלה זו התקבלו שני פתרונות:

1. *<u>ללא אופטימיזציות</u>*
נבצע אנליזת חיות על המשתנים שלנו ונקבל:
- המשתנה a חי החל משורה 2 ועד שורה 8
- המשתנה b חי מתחילת הקוד ועד שורה 8
- המשתנה c חי מתחילת הקוד (עבור המקרה בו התנאי בקפיצה המותנית משוערך לfalse) ועד שורה 9
- המשתנה d חי משורה 3 ועד שורה 9
- המשתנה e מוגדר בשורה 8 אבל לא חי לאחריה. נצטרך להקצות לו רגיסטר לצורך החישוב אך נוכל לשחררו מיד לאחר סיום החישוב

- המשתנה f חי משורה 9 עד שורה 10

נשים לב למשל שאם המשתנה a חי עד שורה 8 והמשתנה e מוגדר בשורה 8, אזי שניהם יכולים להשתמש באותו רגיסטר. אינטואיטיבית, נבחין שבכל נקודה בקוד חיים לכל היותר 4 משתנים. פורמלית, לפי הנתונים הללו, נוכל לבנות interference graph עבור המשתנים שלנו:



לא ניתן לצבוע את הגרף עם 3 צבעים, אך כן ניתן לצבוע אותו עם 4 צבעים. לכן התשובה היא שלא ניתן להסתפק ב3 רגיסטרים ונצטרך לכל הפחות 4 רגיסטרים.

2. *<u>עם אופטימיזציות</u>*

לאחר הפעלת האופטימיזציות שראינו בכיתה נקבל את הקוד הבא:

```
1.  void foo() {
2.      a := b + 1
3.      if a < 10 goto 5
4.      c := 15
5.      f := c * 5
6.      print f
7.  }
```

נשים לב שכעת נותרו לנו רק 4 משתנים ולפי אנליזת חיות נקבל:

- המשתנה a חי החל משורה 2 ועד שורה 3
- המשתנה b חי מתחילת הקוד ועד שורה 2
- המשתנה c חי מתחילת הקוד (עבור המקרה בו התנאי בקפיצה המותנית משוערך לfalse) ועד שורה 5
- המשתנה f חי משורה 5 עד שורה 6

הinterference graph המתקבל עבור הקוד הזה הוא:

את הגרף הזה ניתן לצבוע עם שני צבעים, ולכן 3 רגיסטרים יספיקו.

ג. (5 נק') נתון כי הקומפיילר תמיד מקצה רגיסטרים למשתנים לפי הסדר הלקיסקלי של המשתנים. כמו כן, נתון שהקומפיילר תמיד יעדיף להקצות את הרגיסטר הנמוך ביותר הפנוי (כלומר r0 לפני r1, r1 לפני r2 וכך הלאה). הניחו כי לא קיימת הגבלה על מספר הרגיסטרים הקיימים במעבד. במקרה בו אף משתנה אינו חי בסוף המתודה, וללא שימוש באופטימיזציות נוספות, הראו מה תהיה הקצאת הרגיסטרים המינימלית למשתנים (מינימלית במספר הרגיסטרים בהם משתמשים).

נתבסס על המקרה הראשון מסעיף ב (ללא אופטימיזציות). אנו יודעים כבר שנצטרך להשתמש ב4 רגיסטרים. נעבור על הinterference graph בסדר לקסיקלי (נתחיל מa, אחריו b, אחריו c, וכך הלאה) ונקצה בכל פעם את הרגיסטר הפנוי הנמוך ביותר:

- המשתנה a יישמר ברגיסטר r0
- המשתנה b יישמר ברגיסטר r1

- המשתנה c יישמר ברגיסטר r2
- המשתנה d יישמר ברגיסטר r3
- המשתנה e יישמר ברגיסטר r0 (רגיסטר זה פנוי כי a כבר לא חי)
- המשתנה f יישמר ברגיסטר r0 (רגיסטר זה פנוי כי e כבר לא חי)

בבדיקת השאלה התקבלו גם פתרונות שלא עמדו בדרישה להקצאה בסדר לקסיקלי של המשתנים.

חלק ב (10 נקודות) – ייצור קוד עבור תכנות מונחה עצמים

נתונות המחלקות A ו B הבאות:

```
class A {
        int x;
        int y;
}

class B : public A {
        int z;
}

void f ( A a ) {
        print( a.x );
}

void g ( B b ) {
        print ( b.y + b.z );
}
```

סטן הציע להחזיק אובייקטים מסוג B בזיכרון כך שהשדות של B ממוקמים לפני השדות הנורשים מA. כלומר, אובייקט מטיפוס B יראה בזיכרון כך:

B

z : int
x : int
y : int

a. (7 נק') האם פתרון זה יעבוד? שימו לב להתייחס בתשובתכם לאופן הגישה לשדות וקריאה למתודות.

נשים לב שהפתרון המוצע מתייחס אך ורק לסדר בין שדות האובייקט ומבנה האובייקט בזכרון. הוא אינו מתייחס לאופן הפעולה של הקומפיילר מעבר לכך. כמו כן, המחלקות בדוגמה לא מכילות מתודות כלל, בפרט לא מתודות וירטואליות, לכן הדוגמה לא כוללת מצביע לvtable.
מעשית, סידור הזכרון באופן הזה הוא פישוט של הצורה בה נשמרים אובייקטים בזכרון במקרה של ירושה מרובה (multiple inheritance, שהוזכר בהרצאה משבוע 11 של הסמסטר).
פתרון כזה בהחלט יכול לעבוד אך נצטרך לעדכן את ייצור הקוד בקומפיילר כדי לתמוך בו.
עבור קריאה של מתודה שמצפה לקבל אובייקט מטיפוס A (למשל המתודה f) עם ארגומנט מטיפוס A, אין צורך לבצע שינויים. כנ"ל עבור מתודה שמצפה לקבל ארגומנט מטיפוס B (למשל g). עבור גישה לשדות של A מאובייקט מטיפוס A וגישה לשדות של A או B מאובייקט מטיפוס B גם כן אין צורך בשינויים.
השינוי היחיד הנדרש הוא כאשר קוראים למתודה שמצפה לאובייקט מטיפוס A עם ארגומנט מטיפוס B (או טיפוס שיורש מA במקרה הכללי). נזכור שהקומפיילר יודע את מבנה המחלקות, יודע איזה שדות הם מכילים, יודע את טיפוסי הארגומנטים המצופים בכל מתודה ויודע את הטיפוס המוגדר של כל משתנה (נבדיל בין הטיפוס של המשתנה בזמן ריצה לבין הטיפוס הסטטי שלו בקוד, לצורך התמיכה במבנה הזכרון המבוקש מספיק הטיפוס הסטטי).
נפצל לשני מקרים:

- אובייקטים מועברים כארגומנט by-value:
כאשר נעשה reduce לכלל דקדוק של קריאה למתודה, נצטרך לייצר קוד שמעתיק את ערכי שדות האובייקט לארגומנט. במקרה הזה הקומפיילר יעתיק לארגומנט רק את השדות שמשותפים לA ולB.
- אובייקטים מועברים כארגומנט by-reference (כלומר מועברת כתובת האובייקט בזכרון):
במקרה זה במקום להעביר את כתובת האובייקט B, נקדם את המצביע כך שיצביע לתחילת האובייקט A ונעביר את כתובת זו.
כלומר, בהנתן אובייקט b מטיפוס B, במקום להעביר את &b נעביר (&b+sizeof(fields of B.

b. (3 נק') האם תשובתכם לסעיף א' תשתנה אם נגדיר שהשדות נשמרים בסדר הפוך להגדרתם, כלומר הסדר בין השדות נשמר ואובייקט מטיפוס B יראה בזיכרון כך:

B

z : int
y : int
x : int

קודם כל, הפתרון המוצע בסעיף א יעבוד גם בסעיף הזה. מאחר ואנו יודעים מראש השדות בכל האובייקטים ישמרו בסדר הפוך, הoffsetים בטבלת הסמלים יחושבו בהתאם ונוכל לבצע גישה לשדות באותו אופן כפי שעשינו תמיד.

בסעיף זה התאפשרו פתרונות נוספים המבוססים על הסדר ההפוך בו נשמרים השדות. פתרונות אלה מבוססים על גישה לאובייקטים מסוף האובייקט (במקום מתחילת האובייקט כפי שעשינו עד כה). בפתרונות הללו נדרש גם לציין איך נדע את סוף האובייקט מאחר וזה מידע לא זמין לנו בזמן ריצה, לא ניתן לחישוב (בהנחה שאנו לא יודעים מראש את הטיפוס הדינמי של האובייקט) ובלעדיו לא נוכל לממש את הפתרון. פתרונות שהתקבלו הם:

- בעת הקצאת אובייקט אנו יודעים מה הטיפוס הדינמי שלו (הטיפוס המוקצה). לכן כאשר נעשה reduce לכלל דקדוק שמקצה אובייקט, נייצר קוד שבמקום לשמור את כתובת תחילת האובייקט שומר את כתובת סוף האובייקט (תחילת האובייקט + גודל הטיפוס). כך מעשית הפכנו את כל כיוון העבודה עם אובייקטים (נצטרך לעדכן בהתאם את כל המקומות בהם אנו מייצרים קוד שמשתמש באובייקטים, אך הפתרון יעבוד).
- נוסיף שדה נוסף בתחילת כל אובייקט שמחזיק את גודל האובייקט. נאתחל שדה זה בקוד שנייצר בreduce של כלל דקדוק שמקצה אובייקט. פתרון זה חורג ממבנה הזכרון שהוגדר בשאלה, אך מאחר והוא משמר את סדר השדות בזכרון החלטנו לקבלו.