28.01.2019

מבחן סוף סמסטר – מועד א׳

מרצה אחראי: ד״ר שחר יצחקי

מתרגלים: יעקב סוקוליק, אלכסנדר סיבק, עומר כץ

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 נקודות)

נתון קטע הקוד הבא בשפת FanC:

```
1.  struct s {
2.      int x;
3.      int y;
4.  };
5.
6.  bool foo(int a, struct s c) {
7.      if (a > 0)
8.          return c.x < c.y;
9.      else {
10.         int d = 8;
11.         return false;
12.     }
13. }
14.
15. void main() {
16.     struct s a;
17.     a.x = 1;
18.     a.x = a.x + 1;
19.     a.y = 2;
20.     bool res = foo(0, a);
21.     return;
22. }
```

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 3 בשורה הבאה:

```
3.      byte y;
```

ב. מחליפים את שורה 18 בשורה הבאה:

```
18.     a.x++;
```

ג. מחליפים את שורה 11 בשורה הבאה:

```
11.     return 0;
```

ד. מחליפים את שורה 10 בשורה הבאה:

```
10.         int d = 8 / a;
```

ה. מחליפים את שורה 10 בשורה הבאה:

```
10.         int d_1 = 8;
```

חלק ב – הרחבת השפה (10 נקודות)

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 את היכולת להגדיר פונקציות אנונימיות.
פונקציות אנונימיות אינן מחוברות באופן חח״ע למזהה יחיד. ההגדרה של פונקציה כזאת תהיה באמצעות התחביר:

```
Func myFunc =  int @(int x, int y) {  return  x*y; }
```

לאחר מכן, נוכל להשתמש בפונקציות האלה באופן הבא, לדוגמא:

```
Func anotherOne = myFunc;
printi(anotherOne(2, 5));  // Prints 7
```

לאחר הגדרה זו, ניתן להשתמש ב-myFunc או ב- anotherOne אם הם הוגדרו ב-scope הנוכחי או באב קדמון שלו. יש לבצע את הבדיקות לתקינות הטיפוסים המועברים.
לא ניתן להעביר את הפונקציות האנונימיות כארגומנטים לפונקציות אחרות.
לא ניתן להכריז על משתנה מטיפוס Func מבלי לקבוע את תוכנו. לא ניתן לשנות את ערכו של משתנה מטיפוס Func לאחר הגדרתו.
הפונקציות האנונימיות אינן יכולות להשתמש במשתנים החיצוניים להגדרת הפונקציה – כל המשתנים חייבים להיות לוקאליים לפונקציה האנונימית. אין קינון של פונקציות אנונימיות.

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.

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
```

א. (5 נק׳) עזרו לקרטמן לתקן את האנליזה שלו כך שתהיה נאותה.

ב. (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.

שאלה 3 (10 נקודות): אופטימיזציות

להלן קטע קוד מתוכנית בשפת C:

```
1. void main() {
2.    int f = 3;
3.    int a = 3*f;
4.    int b = a + secretSauce();
5.    if (b > 300) {
6.        printf("A%d\n", a + secretSauce());
7.    }
8.    else {
9.       if (a < 30000) {
10.           int k = 0;
11.           while (k < 30000) {
12.               k += a;
13.           }
14.           printf("B%d", k);
15.       }
16.       else {
17.           printf("C%d", 3*f+secretSauce());
18.       }
19.   }
20.   printf("D%d", 3*f+b-secretSauce());
21.}
```

א. (6 נק') בצעו עבור קטע הקוד הנ"ל את האופטימיזציות הבאות ככל הניתן:

- Constant Propagation
- Copy Propogation
- Constant Folding
- Common sub-expression elimination
- Algebraic Simplification
- Useless code elimination

רשמו את מספרי השורות שבהן ביצעתם את השינויים, בתוספת השינוי שביצעתם ואת סוג האופטימיזציה שגרם לכך.

ב. (4 נק') הציעו אופטימיזציה נוספת, המבוססת על אנליזת DFA, שעלולה להביא לשיפור בזמן הריצה. הניחו כי קוד המתודה secretSauce ידוע לנו. פרטו בקצרה והסבירו מתי היא תביא לשיפור.

שאלה 4 (15 נק׳): ניתוח תחבירי וסמנטי

נתון הדקדוק הבא המייצג מילים המכילות סוגריים מאוזנים ומספרים. הדקדוק הוא מעל האסימונים lpar (סוגריים שמאליים), rpar (סוגריים ימניים) ו-num (מספר). אסימונים מופיעים בדקדוק עם קו תחתון, משתנים ללא קו תחתון:

1. $S \rightarrow L$
2. $L \rightarrow \underline{lpar}\ L\ \underline{rpar}\ L$
3. $L \rightarrow \underline{num}$
4. $L \rightarrow \epsilon$

הניחו כי לאסימון num קיימת התכונה הסמנטית val המכילה את ערך המספר.

א. (6 נק׳) מה היא מחלקת דקדוקי LR הפשוטה ביותר אליה משתייך הדקדוק הנתון? הוכיחו.

נרצה שכל מילה בשפה תקיים את התכונה הסמנטית הבאה: כל מספר המופיע במילה מייצג את מספר הסוגריים השמאליים הפתוחים.
לדוגמה, המילים הבאות תקינות סמנטית:

$$((2)(2))$$
$$()\ 0$$

ואילו מילים אלו אינן תקינות סמנטית:

$$((1)(2))$$
$$((2)(2))\ 1$$

ב. (6 נק׳) כתבו כללים סמנטיים לבדיקת התכונה הסמנטית לעיל בזמן ניתוח. הבדיקה תתבצע באמצעות קריאה לפונקציה check שחתימתה היא: void check(bool b)
הפונקציה check משערכת את הארגומנט ובמידה וערכו הוא false, היא מסיימת את ריצת המנתח. יש לדווח על השגיאה מוקדם ככל הניתן.
השתמשו **בתכונות נורשות 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.

ג. (3 נק׳) פתרו את סעיף ב׳ באמצעות שימוש **בתכונות נוצרות בלבד**.

שאלה 5 (25 נקודות): Code Generation

חלק א (15 נקודות) - Register Allocation

נתון המתודה הבאה:

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

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.

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

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).

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

נתונות המחלקות A וB 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

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

ב. (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

בהצלחה!

נוסחאות ואלגוריתמים

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

Top Down

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

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

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

הגדרה: דקדוק G הוא LL(1) אם ורק אם 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:

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

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:

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

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

39. $ExpList \rightarrow Exp$
40. $ExpList \rightarrow Exp\ COMMA\ ExpList$
41. $Type \rightarrow INT$
42. $Type \rightarrow BYTE$
43. $Type \rightarrow BOOL$
44. $StructType \rightarrow STRUCT\ ID$
45. $Exp \rightarrow LPAREN\ Exp\ RPAREN$
46. $Exp \rightarrow Exp\ BINOP\ Exp$
47. $Exp \rightarrow ID$
48. $Exp \rightarrow ID\ PERIOD\ ID$
49. $Exp \rightarrow Call$
50. $Exp \rightarrow NUM$
51. $Exp \rightarrow NUM\ B$
52. $Exp \rightarrow STRING$
53. $Exp \rightarrow TRUE$
54. $Exp \rightarrow FALSE$
55. $Exp \rightarrow NOT\ Exp$
56. $Exp \rightarrow Exp\ AND\ Exp$
57. $Exp \rightarrow Exp\ OR\ Exp$
58. $Exp \rightarrow Exp\ RELOP\ Exp$