4.7.2018

מבחן סוף סמסטר – מועד א'

מרצה אחראי: ד"ר אוהד שחם

מתרגלים: אבנר אליזרוב, עומר כץ, הילה פלג

הוראות:

א. בטופס המבחן 11 עמודים מהם 5 דפי נוסחאות. בדקו שכל העמודים ברשותכם.

ב. משך המבחן שלוש שעות (180 דקות).

ג. אסור כל חומר עזר חיצוני.

ד. במבחן 6 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.  int foo(byte i) {
2.      bool a[5];
3.      a[i] = true;
4.  }
5.  void main() {
6.      byte i = 0b;
7.      while ( i < 4b ) {
8.          i = i +2b;
9.          foo(i);
10.     }
11. }
```

בסעיפים הבאים מוצגים שינויים (בלתי תלויים) ל-main 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:

1. מחליפים את שורה 6 בשורה הבאה:

```
6. int i = 0b;
```

2. מחליפים את שורה 8 בשורה הבאה:

```
8. i = i + 3;
```

3. מחליפים את שורה 2 בשורה הבאה:

```
2. bool a[4.0];
```

4. מחליפים את שורה 7 בשורה הבאה:

```
7. while ( i < 2 ) {
```

5. מחליפים את שורה 8 בשורה הבאה:

```
8. i++;
```

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**.

שפת FanC מאפשרת המרת טיפוסים מובלעת (implicit) בהשמה והעברת פרמטרים לפונקציה. נרצה לאפשר גם המרה מפורשת (explicit) על ידי אופרטור המרת טיפוסים בדומה ל-C. כך ניתן, למשל, יהיה להמיר ערך בוליאני למספרי כך:

```
bool x = true;
int y = (int)x;
```

או מספרי לבוליאני כך:

```
int cond = 18;
if ((bool)cond) {
     //…
}
```

וכמובן, תתאפשר המרה בין טיפוסים מספריים.
המרה בין שני טיפוסים מערכים, בין אם בגודל שונה, או מטיפוס איברים שונה, תמשיך להיות אסורה.
ערכו של ביטוי אחרי המרה יהיה מטווח הערכים המותר של טיפוס היעד. בהמרה לביטוי בוליאני, ערך מספרי שאינו 0 יקבל את הערך true וערך 0 את הערך false. בהמרה מבוליאני למספר, הערך true יקבל את הערך 1 והערך false את הערך 0. בהמרות בין טיפוסים חשבוניים, יבוצע truncation (איפוס הביטים הגבוהים) אם יש צורך להתאים את המספר לטווח הערכים של טיפוס היעד.

<u>שאלה 2 (30 נקודות): אופטימיזציות</u>

נתון קטע הקוד הבא:

```
1. if (x < 4) {
2.     print ("X is smaller than 4");
3. }
4.
5. if (x < 10) {
6.     print ("X is smaller than 10");
7. }
```

או בשפת ביניים:

```
1. if x >= 4  goto 3
2. print ("X is smaller than 4");
3. if x >= 10 goto 5
4. print ("X is smaller than 10");
5. nop
```

א. (3 נק') בנה את ה CFG של קטע הקוד.

ב. (10 נק') הסבר איזה אופטימיזציה אפשרית, מה היא חוסכת וצייר את ה CFG לאחר האופטימיזציה. שים לב כי האופטימיזציה הנדרשת מורידה זמן ריצה.

ג. (12 נק') הצע אנליזה המוצאת צמתים ב CFG בהם ניתן לבצע את האופטימיזציה מסעיף ב. הסבר כיצד תמצא את הצמתים. ציין באיזו אנליזות הנך צריך להשתמש ואיזה תנאיי נכונות צריך לבדוק ולשמר. שים לב כי האנליזה צריכה לעבוד על תוכנית כללית כלשהי ולא רק על הדוגמא הספציפית.

ד. (5 נק') האם ניתן היה לבצע את האופטימיזציה אם נשנה את שורה 4 בקוד המקור באופן הבא:

4. print ("Nothing");

הסבר מה צריך היה לבצע ובנה את ה CFG של התוכנית לאחר האופטימיזציה.

שאלה 3 (10 נקודות): DFA

נוסיף לשפת הביניים פקודה חדשה בשם flush x שמפעפעת את ערך המשתנה x מה-cache לזיכרון הראשי. לכל נקודה **p** בתכנית נאמר כי משתנה x יציב אם עד נקודה **p** הופיעה בהכרח פקודת flush x לאחר ההשמה האחרונה ל-x או שלא התבצעה השמה ל-x כלל.

לדוג', עבור קטע הקוד הבא שמכיל את המשתנים b ו-c -

1. b:=0
2. c:=3
3. flush b
4. flush c
5. b:=c
6. if(b>0) goto 8
7. flush b
8. print c

לפני שורה 1, כל המשתנים יציבים.
לפני שורה 2, רק c יציב.
לפני שורה 3, אף משתנה אינו יציב.
לפני שורה 4, רק b יציב.
לפני שורה 5 כל המשתנים יציבים.
לפני שורה 8 רק c יציב.

הראו אנליזת DFA המחשבת לכל בלוק בסיסי את כל המשתנים **היציבים** לפניו.

הערות:

- ניתן להגדיר שורה בודדת כבלוק בסיסי, אך יש לציין זאת.
- ניקוד מלא יינתן על פתרון יעיל שאינו דורש מעברים מיותרים על פריטי המידע.
- יורדו נקודות על אי הגדרה מלאה של DFA. כלומר, יש לציין מהם פריטי המידע, מהו סוג האנליזה, כיוון האנליזה, פונקציית האנליזה, משוואת הזרימה, והאתחול. כמו כן, יש להסביר כיצד ניתן לקבל את הפתרון לשאלה מתוצאות האנליזה.

שאלה 4 (15 נק'): ניתוח תחבירי וסמנטי

א. נתונה השפה הבאה – {aa, ab}
   a. (5 נקודות) כתבו דקדוק עבור השפה כך שהדקדוק אינו LL(1) אך כן LR(0). הוכיחו כי הוא אינו LL(1) והוא כן LR(0).
   b. (5 נקודות) כתבו דקדוק עבור השפה כך שהדקדוק הינו LL(1). הוכיחו כי הוא LL(1).

ב. (5 נקודות) סטודנט שחוזר על הקורס טוען שאין צורך במחסנית ה- offsets עבור טבלת הסמלים וניתן להשתמש במשתנה יחיד מסוג int (עדיין קיימת מחסנית עבור הטבלאות עצמן). הציעו דרך שבה ניתן יהיה להשתמש במשתנה יחיד או הסבירו מדוע שינוי זה אינו אפשרי. אם שינוי זה אפשרי, הסבירו כיצד מתעדכן משתנה ה-offset החדש בפתיחת/סגירת scope ובהוספת משתנה חדש.

שאלה 5 (10 נקודות): רשומות הפעלה

א. (5 נק') נתונה הפונקציה הבאה, ועבור כל תת-ביטוי את הרגיסטרים התפוסים בסיומו:

```
1) int f(int x, byte y, int z) {
2)     if (x < 1) printi(0);
3)     int i = x + 2 + h(y) + g(x-1,y+1,8);
4)     printi(i);
5) }
```

בקוד שיוצר עבור הפונקציה f, ברגע הקריאה לפונקציה g, נמצאים בשימוש הרגיסטרים הבאים:

$s0 מכיל את תוכן הביטוי x+2+h(y)
$t1 מכיל את תוכן הביטוי x-1
$s2 מכיל את תוכן הביטוי y+1
$t3 מכיל את תוכן הביטוי 8

ה-ABI בו השתמש ייצור הקוד עבור הפונקציה f מציין את כל הרגיסטרים כ-caller-save. הציגו מימוש אפשרי ל-g, הכולל את הקצאת הרגיסטרים עבורו, בו החלטת ה-ABI גורמת לכתיבות וקריאות מיותרות מהמחסנית. הסבירו מדוע הן מיותרות.

ב. (5 נק') החליפו את שורה מס' 3 בקוד מסעיף א' בשורה

```
3) int i = x + 2 + h(y) + f(x-1,y+1,8);
```

חישובי הביניים הדורשים רגיסטר הם אלה המופיעים בסעיף א'.

האם ניתן לממש את הקצאת הרגיסטרים של f כך שיחסכו כתיבות למחסנית?

שאלה 6 (15 נקודות): Backpatching

נתון כלל הדקדוק עבור מבנה הקרה parallel_loop:

$S \rightarrow parallel\_loop\ (B\_List)\ S\_List;$

$S\_List \rightarrow S;\ S\_List_1 \mid S$

$B\_List \rightarrow B;\ B\_List_1 \mid B$

נניח ש B_List מייצג רשימה של תנאים בוליאניים מסוג B וS_List רשימה של משפטים מסוג S. נניח שהאורך של S_List זהה לאורך של B_List (אין צורך לבדוק הנחה זו בקוד).

הלולאה parallel_loop תימשך כל עוד לפחות תנאי אחד מתוך התנאים בB_List מתקיים.

בכל איטרציה של הלולאה, המשפט S_i יתבצע אם"ם התנאי B_i מתקיים.

שימו לב כי יש לבצע את S_i לפני בדיקת התנאי B_{i+1}.

לדוגמא, עבור הקוד:

```
i = 0;
parallel_loop( i < 5; i % 2 == 0; i > 10) i = i+1; print(i); i= 13;
```

תודפס התוצאה:

```
2
4
```

(הפונקציה print בדוגמא הנ"ל מקבלת מספר ומדפיסה אותו למסך ולאחריו מדפיסה newline)

א. (5 נק') הציעו פריסת קוד המתאימה לשיטת backpatching עבור מבנה הבקרה הנ"ל. על הקוד הנוצר להיות יעיל ככל האפשר. הסבירו מהן התכונות שאתם משתמשים בהן עבור כל משתנה.

ב. (10 נק') כתבו סכימת תרגום בשיטת backpatching המייצרת את פריסת הקוד שהצעתם בסעיף הקודם. על הסכימה להיות יעילה ככל האפשר, הן מבחינת זמן הריצה שלה והן מבחינת המקום בזיכרון שנדרש עבור התכונות הסמנטיות.

שימו לב:

- אין לשנות את הדקדוק, למעט הוספת מרקרים N,M שנלמדו בכיתה בלבד.
- אין להשתמש בכללים סמנטיים באמצע כלל גזירה.
- אין להשתמש במשתנים גלובליים בזמן קומפילציה.
- למשתנים S,B ישנן התכונות שהוגדרו בכיתה בלבד.
- למשתנים S,B יש כללי גזירה פרט לאלו המוצגים בשאלה.

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
```

ייצור קוד בשיטת Backpatching

פונקציות:

`makelist(quad)`	יוצרת רשימה ריקה עם איבר אחד (ה"חור" quad).
`merge(list1,list2)`	מחזירה רשימה ממוזגת של הרשימות list1, list2
`emit(code string)`	מדפיסה קוד בשפת הביניים ומאפשרת להדפיס פקודות קפיצה עם "חורים".
`nextquad()`	מחזירה את כתובת הרביעיה (הפקודה) הבאה שתצא לפלט.
`backpatch(list, quad)`	מקבלת רשימת "חורים" list וכתובת quad, ו"מטליאה" את הרשימה כך שבכל החורים תופיע הכתובת quad.
`newtemp()`	מחזירה שם של משתנה זמני חדש שאינו נמצא בשימוש בתכנית.

משתנים סטנדרטיים:

- S: גוזר פקודות (statements) בשפה. תכונות:
  - nextlist: רשימת כתובות של פקודות המכילות חור שיש להטליא בכתובת הפקודה הבאה לביצוע אחרי הפקודה הנגזרת מ- S.
- B: גוזר ביטויים בוליאניים. תכונות:
  - truelist: רשימת כתובות של פקודות המכילות חור שיש להטליא בכתובת אליה יש לקפוץ אם הביטוי הבוליאני מתקיים.
  - falselist: רשימת כתובות של פקודות המכילות חור שיש להטליא בכתובת אליה יש לקפוץ אם הביטוי הבוליאני אינו מתקיים.
- E: גוזר ביטויים אריתמטיים. תכונות:
  - E.place: שם המשתנה הזמני לתוכו מחושב הביטוי האריתמטי.

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`

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

דקדוק:

$Program \rightarrow Funcs$
$Funcs \rightarrow \epsilon$
$Funcs \rightarrow FuncDecl\ Funcs$
$FuncDecl \rightarrow RetType\ ID\ LPAREN\ Formals\ RPAREN\ LBRACE\ Statements\ RBRACE$
$RetType \rightarrow Type$
$RetType \rightarrow\ VOID$
$Formals \rightarrow \epsilon$
$Formals \rightarrow FormalsList$
$FormalsList \rightarrow FormalDecl$
$FormalsList \rightarrow FormalDecl\ COMMA\ FormalsList$
$FormalDecl \rightarrow Type\ ID$
$FormalDecl \rightarrow Type\ ID\ LBRACK\ NUM\ RBRACK$
$FormalDecl \rightarrow Type\ ID\ LBRACK\ NUM\ B\ RBRACK$
$Statements \rightarrow Statement$

$Statements \rightarrow Statements\ Statement$
$Statement \rightarrow LBRACE\ Statements\ RBRACE$
$Statement \rightarrow Type\ ID\ SC$
$Statement \rightarrow Type\ ID\ ASSIGN\ Exp\ SC$
$Statement \rightarrow Type\ ID\ LBRACK\ NUM\ RBRACK\ SC$
$Statement \rightarrow Type\ ID\ LBRACK\ NUM\ B\ RBRACK\ SC$
$Statement \rightarrow ID\ ASSIGN\ Exp\ SC$
$Statement \rightarrow ID\ LBRACK\ Exp\ RBRACK\ ASSIGN\ Exp\ SC$
$Statement \rightarrow Call\ SC$
$Statement \rightarrow RETURN\ SC$
$Statement \rightarrow RETURN\ Exp\ SC$
$Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement$
$Statement \rightarrow IF\ LPAREN\ Exp\ RPAREN\ Statement\ ELSE\ Statement$
$Statement \rightarrow WHILE\ LPAREN\ Exp\ RPAREN\ Statement$
$Statement \rightarrow BREAK\ SC$
$Call \rightarrow ID\ LPAREN\ ExpList\ RPAREN$
$Call \rightarrow ID\ LPAREN\ RPAREN$
$ExpList \rightarrow Exp$
$ExpList \rightarrow Exp\ COMMA\ ExpList$
$Type \rightarrow INT$
$Type \rightarrow BYTE$
$Type \rightarrow BOOL$
$Exp \rightarrow LPAREN\ Exp\ RPAREN$
$Exp \rightarrow ID\ LBRACK\ Exp\ RBRACK$
$Exp \rightarrow Exp\ BINOP\ Exp$
$Exp \rightarrow ID$
$Exp \rightarrow Call$
$Exp \rightarrow NUM$
$Exp \rightarrow NUM\ B$
$Exp \rightarrow STRING$
$Exp \rightarrow TRUE$
$Exp \rightarrow FALSE$
$Exp \rightarrow NOT\ Exp$
$Exp \rightarrow Exp\ AND\ Exp$
$Exp \rightarrow Exp\ OR\ Exp$
$Exp \rightarrow Exp\ RELOP\ Exp$