מס' נבחן

**שם הקורס: קומפילציה**
**קוד הקורס: 10334**

**הוראות לנבחן:**
-חומר עזר שימושי לבחינה:
כל חומר כתוב

-אין לכתוב בעפרון / עט מחיק
-אין להשתמש בטלפון סלולארי
-אין להשתמש במחשב אישי או נייד
-אין להשתמש בדיסק און קי ו/או מכשיר מדיה אחר
-אין להפריד את דפי שאלון הבחינה

בחינת סמסטר: ג
השנה: 2017
מועד: א

תאריך הבחינה: 31/01/17
שעת הבחינה: 17:00
משך הבחינה: 3 שעות

השאלון לא ייבדק בתום הבחינה ע"י המרצה

מרצים: גדי פסח

## מבנה הבחינה והנחיות לפתרון:
**בבחינה 4 שאלות. יש לענות על כולם.**

**שאלה מס' 1 ( 30 נקודות)**
יש לכתוב מפרט ל- bison כך שתתקבל תוכנית כמתואר כאן.
**(המפרט ל- flex נתון והוא מופיע בהמשך).**

התוכנית תקרא את הקלט המתאר רשימה של פרקים של סדרות טלוויזיה ותדפיס את אחוז הסדרות שבהן לפרק הפותח (הפרק הראשון של העונה הראשונה) היו כתוביות בעברית.

עבור כל פרק מופיעים בקלט הנתונים הבאים : שם הסדרה, מספר העונה, מספר הפרק בעונה והשפות בהן יש כתוביות זמינות.

הכתוביות עשויות להיות באחת מהשפות הבאות : אנגלית, עברית, ספרדית ופולנית.
אם לא מופיעות שפות במפורש אז ברירת המחדל היא אנגלית. הניחו שעבור כל פרק יש לכל היותר כתוביות בשתי שפות. פסיקים מפרידים בין השפות.

מספר הפרק מופיע כאות E ולאחריה מספר (למשל E3)
מספר העונה מופיע כאות S ולאחריה מספר (למשל S1)

מספר כלשהו של סימני white space יכולים להפריד בין האסימונים השונים.

בתחילת הקלט מופיעה הכותרת Series.

דוגמא לקלט :

Series
"Orange is the New Black" S1 E1
subtitles: English, Hebrew

"Orange is the New Black" S2 E11
subtitles: English, Hebrew

"Breaking Bad" S3 E9 subtitles: English, Hebrew, Spanish

"Travelers" S1 E1 subtitles: Hebrew, English, Polish

"House Of Cards" S1 E1 subtitles: English, Spanish
"House Of Cards" S5 E7

בדוגמא זו, על התוכנית להדפיס
Percentage of opening episodes with Hebrew subtitles: 66.66

בקלט מופיעים שלושה פרקים פותחים של סדרות מתוכם לשניים יש כתוביות בעברית.
שימו לב שרק פרקים פותחים (פרק ראשון בעונה ראשונה) נלקחים בחשבון.

**יש לכתוב מפרט ל- bison** . יש לרשום את כללי הגזירה של הדקדוק ביחד עם הפעולות הסמנטיות המתאימות.
יש לכתוב גם את ההכרזות הנחוצות (אם הן נחוצות) מסוג %union, %token, %type.

כמובן שהמפרט של bison צריך להתאים למפרט הנתון של flex.
אין להגדיר משתנים גלובליים בפתרון (בנוסף לאלו המוגדרים ע"י flex ו- bison).

הניחו שמוגדר enumeration type כזה:

```
enum language { HEBREW, ENGLISH, SPANISH, POLISH };
```

**הנה המפרט ל- flex:**
(הערה: הפונקציה atoi ממירה מחרוזת למספר מטיפוס int. לדוגמא atoi ("7") מחזירה 7 כמספר מטיפוס int).

```
%%
"Series"        { return SERIES; }
"subtitles"     { return SUBTITLES; }


S[0-9]+         { yylval.num = atoi (&yytext[1]);
                  return SEASON_NUMBER;

E[0-9]+         { yylval.num = atoi (&yytext[1]);
```

```
                return EPISODE_NUMBER;

[:,]            { return yytext[0];}

\"[A-Z][a-z]*(" "[A-Za-z]*)*\"     {
                return NAME; }


"English" { yylval.lang = ENGLISH;
            return LANGUAGE; }

"Hebrew" { yylval.lang = HEBREW;
            return LANGUAGE; }

"Spanish" { yylval.lang = SPANISH;
            return LANGUAGE; }

"POLISH" { yylval.lang = POLISH;
            return LANGUAGE; }

[ \t\n]+                    /* skip white space */

.       { fprintf (stderr, "illegal token: %c",
                yytext [0]); exit (1); }
```

**שאלה מס' 2 (25 נקודות)**
**סעיף א' (17 נקודות)**

Construct the SLR (1) table for the following grammar.
Also draw the diagram for the finite automaton which you constructed in order to build the table.

```
(1) S    z A
(2) S → z B
(3) A → a c
(4) B → epsilon
```

**סעיף ב' (5 נקודות)**

Show the steps a parser (using the SLR (1) table you constructed) will do when parsing the input z a c $.

Complete the following table:

| stack contents | remaining input | action |
|---|---|---|
| 0 | z a c $ | |

.
Write shift or reduce or accept in every entry in the action column. In case the action is reduce – also write the production used in the reduction.

**סעיף ג: (3 נקודות)**

What is the maximum number of shifts a parser using the SLR(1) table you constructed will do when processing a legal input (a word which can be derived from the grammar mentioned above) ?

**שאלה מס' 3 (15 נקודות)**

סעיף א (10 נקודות)

בנו טבלת LL(1) עבור הדקדוק הבא:

```
(1) S -> a A c
(2) A -> B G a
(3) B -> b
(4) B -> epsilon
(5) G -> g
(6) G -> epsilon
```

**סעיף ב (5 נקודות)**

הראו את הריצה של ה- parser המשתמש בטבלה של סעיף א על הקלט aac

השלימו את הטבלה הבאה:

| כלל גזירה בו נעשה שימוש | יתרת הקלט | תוכן המחסנית |
|---|---|---|
| | aac$ | S$ |

**שאלה מס' 4 (10 נקודות)**

השאלה עוסקת באלגוריתם לתכנות דינמי שמטרתו ליצר קוד אופטימאלי עבור מעבד.
המעבד כולל שני רגיסטרים r0 ו- r1.
סוגי פקודות המעבד שרלוונטיות כאן הן :

```
(1) DIV r r r
(2) DIV r r M
```

כאן r מציין רגיסטר ו- M מציין אופרנד שנמצא בזכרון.
הפקודות האלו מבצעות חילוק של שני אופרנדים וכותבות את התוצאה לרגיסטר היעד (רגיסטר היעד הוא הרגיסטר השמאלי ביותר המופיע בפקודה) . ההבדל בין הפקודות הוא במיקום של שני האופרנדים: בפקודה (1) שניהם ברגיסטרים, בפקודה (2) האופרנד הימני בזכרון והאופרנד השמאלי ברגיסטר.
שימו לב שפקודה (1) משתמשת בשני רגיסטרים בעוד שפקודה (2) משתמשת ברגיסטר אחד בלבד. בשני המקרים האופרנד השמאלי נמצא ברגיסטר שמשמש גם כרגיסטר היעד.

למעבד יש גם פקודה STORE M r שמעתיקה רגיסטר לזכרון. זו הפקודה היחידה של

המעבד שמאפשרת כתיבה לזכרון .

המחיר של פקודת DIV מסוג (1) הוא 4. המחיר של פקודת DIV מסוג (2) הוא 6. המחיר של פקודת ה- STORE הוא 5.

נניח שמפעילים את האלגוריתם לתכנות דינאמי כדי לייצר קוד עבור ביטוי מסוים.
אחד מצמתי עץ הביטוי (נקרא לצומת N) מסומן באופרטור החילוק . וקטור המחירים של הילד השמאלי של N הוא (18 , 17 , 13) כאשר 18 המחיר של חישוב הביטוי לתוך הזכרון, 17 המחיר של חישוב הביטוי לתוך רגיסטר כאשר לרשותנו עומד רגיסטר אחד ו- 13 המחיר של חישוב הביטוי לתוך רגיסטר כאשר לרשותנו עומדים שני רגיסטרים.

וקטור המחירים של הילד הימני של N הוא (8 , 3 , 3).

**חשבו את הרכיב הימני ביותר עבור הצומת N.** ציינו גם באיזה משני סוגי הפקודות (1 או 2) יש להשתמש כדי להשיג מחיר זה. במקרה של שימוש בפקודה מסוג 1 ציינו גם איזה אופרנד יש לחשב קודם: את הימני או את השמאלי.

**<u>שאלה מס' 5 (20 נקודות)</u>**

השאלה עוסקת בשימוש ב- bison לצורך תרגום לקוד ביניים של לולאות מסוג חדש -- לולאות double while.

הנה כלל גזירה המתאר לולאות מסוג זה

```
stmt :
   DOUBLE_WHILE  '(' expression ')' stmt '(' expression ')' stmt
```

הלולאה שקולה לקוד הבא

```
while (expression1)
   stmt1
while (expression2)
   stmt2
```

(כאן expression1 הוא הביטוי הראשון, expression2 הוא הביטוי השני ודבר דומה נכון עבור ה- stmts).

ביטוי שערכו הוא אפס נחשב false וכל ביטוי אחר נחשב true.

**נתבונן בדוגמא:**

```
double_while
  (a + 100)
     a = (3 * a) - c;
  (b + 7 - y)
     b = 5 * z;
```

ניתן לתרגם משפט זה לקוד ביניים באופן הבא. ליד כל פקודה מופיע מספרה (מספרי הפקודות אינן חלק מהקוד). בדוגמא נניח שקטע הקוד מתחיל בשורה 100.

```
100:  t1 = a + 100
101:  if a == 0 goto 106
102:  t2 = 3 * a
103:  t3 = t2 - c
104:  a = t3
```

```
105:  goto 100

106:  t4 = b + 7
107:  t5 = t4 - y
108:  if t5 == 0 goto 112
109:  t6 = 5 * z
110:  b = t6
111:  goto 106
```

**יש להוסיף לכלל הגזירה הנ"ל פעולות סמנטיות (בפסאודו קוד) הנחוצות לצורך התרגום לקוד ביניים.**

בפתרון יש להניח שפקודות בקוד הביניים נצברות בתוך טבלה. המשתנה הגלובלי next_instr מחזיק את האינדקס של הכניסה הפנויה הבאה בטבלה. הפונקציה emit () יוצרת פקודה בקוד ביניים, מוסיפה אותה לטבלה (לכניסה next_instr) ומקדמת את next_instr באחד.

אפשר להשתמש בפונקציות הבאות:
הפונקציה makelist () יוצרת רשימה חדשה של אינדקסים (לתוך הטבלה הנ"ל), מכניסה לתוכה את האינדקס אותה היא מקבלת כארגומנט ומחזירה את הרשימה שיצרה.

הפונקציה merge () מקבלת שתי רשימות של אינדקסים (לתוך הטבלה הנ"ל) ומחזירה רשימה שהיא המיזוג של שתיהן.

הפונקציה backpatch () מקבלת שני ארגומנטים. הראשון הוא רשימה של אינדקסים של פקודות קפיצה שיעדן נותר ריק (הארגומנט הראשון יכול גם להיות אינדקס של פקודת קפיצה אחת). הארגומנט השני הוא האינדקס של היעד. היא עוברת על הפקודות ברשימה (שנמצאות בטבלה) וממלאת את יעד הקפיצה שלהן באינדקס היעד.

לפונקציות הנ"ל ניתן להעביר כארגומנט גם את הערך NULL המייצג בהקשר זה רשימה ריקה של אינדקסים.

הערך הסמנטי של expression הוא המשתנה שיחזיק (בזמן ריצת התוכנית) את תוצאת חישוב הביטוי. .

מותר להוסיף לדקדוק משתנים חדשים שגוזרים את המילה הריקה למשל:

```
marker :   /* empty */    {  $$ = next_instr; }
```



# בהצלחה!

# פתרון

פתרון מועד ג'
10334

# פתרון מבחן בקומפילציה קיץ 2017 (מבחן ראשון)

## שאלה מס' 1

יש לכתוב מפרט ל- bison כך שתתקבל תוכנית כמתואר כאן.
**(המפרט ל- flex נתון והוא מופיע בהמשך).**

התוכנית תקרא את הקלט המתאר רשימה של פרקים של סדרות טלוויזיה ותדפיס את אחוז הסדרות שבהן לפרק הפותח (הפרק הראשון של העונה הראשונה) היו כתוביות בעברית.

עבור כל פרק מופיעים בקלט הנתונים הבאים: שם הסדרה, מספר העונה, מספר הפרק בעונה והשפות בהן יש כתוביות זמינות.

הכתוביות עשויות להיות באחת מהשפות הבאות: אנגלית, עברית, ספרדית ופולנית.
אם לא מופיעות שפות במפורש אז ברירת המחדל היא אנגלית. הניחו שעבור כל פרק יש לכל היותר כתוביות בשתי שפות. פסיקים מפרידים בין השפות.

מספר הפרק מופיע כאות E ולאחריה מספר (למשל E3)
מספר העונה מופיע כאות S ולאחריה מספר (למשל S1)

מספר כלשהו של סימני white space יכולים להפריד בין האסימונים השונים.

בתחילת הקלט מופיעה הכותרת Series.

דוגמא לקלט:

```
Series
"Orange is the New Black" S1 E1
   subtitles: English, Hebrew

"Orange is the New Black" S2 E11
   subtitles: English, Hebrew

"Breaking Bad" S3 E9  subtitles:  English, Hebrew,

"Travelers"  S1 E1 subtitles: Hebrew, English

"House Of Cards"  S1 E1 subtitles: English, Spanish
"House Of Cards" S5 E7
```

בדוגמא זו, על התוכנית להדפיס

```
Percentage of opening episodes with Hebrew subtitles:
66.66
```

בקלט מופיעים שלושה פרקים פותחים של סדרות מתוכם לשניים יש כתוביות בעברית.
שימו לב שרק פרקים פותחים (פרק ראשון בעונה ראשונה) נלקחים בחשבון.

**יש לכתוב מפרט ל- bison** . יש לרשום את כללי הגזירה של הדקדוק ביחד עם הפעולות הסמנטיות המתאימות.
יש לכתוב גם את ההכרזות הנחוצות (אם הן נחוצות) מסוג %union, %token, %type.

כמובן שהמפרט של bison צריך להתאים למפרט הנתון של flex.
אין להגדיר משתנים גלובליים בפתרון (בנוסף לאלו המוגדרים ע"י flex ו- bison).

הניחו שמוגדר enumeration type כזה:

```
enum language { HEBREW, ENGLISH, SPANISH, POLISH };
```

**הנה המפרט ל- flex:**

(הערה: הפונקציה atoi ממירה מחרוזת למספר מטיפוס int. לדוגמא atoi ("7") מחזירה 7 כמספר מטיפוס int).

```
%%
"Series"       { return SERIES; }
"subtitles"    { return SUBTITLES; }


S[0-9]+       { yylval.num = atoi (&yytext[1]);
                return SEASON_NUMBER;

E[0-9]+       { yylval.num = atoi (&yytext[1]);
                return EPISODE_NUMBER;

[:,]           { return yytext[0];}

\"[A-Z][a-z]*(" "[A-Za-z]*)*\"      {
                return NAME; }


"English" { yylval.lang = ENGLISH;
            return LANGUAGE; }

"Hebrew" { yylval.lang = HEBREW;
           return LANGUAGE; }

"Spanish" { yylval.lang = SPANISH;
            return LANGUAGE; }

"POLISH" { yylval.lang = POLISH;
           return LANGUAGE; }

[ \t\n]+                /* skip white space */

.      { fprintf (stderr, "illegal token: %c",
                  yytext [0]); exit (1); }
```

<u>פתרון</u>

```
%union {
    int num;
    enum language lang;
    int has_hebrew_subtitles;
    int isOpeningEpisode;
    struct {
       int num_of_oe; /* number of opening episodes */
       int num_of_hebrew_oe; /* number of opening
episodes having Hebrew subtitles. */
    } oe;


}

%token SERIES SUBTITLES NAME
%token <num> SEASON_NUMBER EPISODE_NUMBER
%token <lang> LANGUAGE

%type <oe> elist episode
%type <isOpeningEpisode> e_spec
%type < has_hebrew_subtitles> subtitle_spec

%%
start:  SERIES elist
             {
               if ($2.num_of_oe != 0)
                 print("Percentage of opening episodes
                        with Hebrew subtitles: %f\n",
                      $2.num_of_hebrew_oe/$2.num_of_oe);
             }

elist:  elist episode {
          $$.num_of_oe = $1.num_of_oe +
                         $2.num_of_oe;
          $$.num_of_hebrew_oe = $1.num_of_hebrew_oe +
                                $2.num_of_hebrew_oe;
        }

elist: /* empty */ { $$.num_of_oe = 0;
                     $$.num_of_hebrew_oe = 0;
                   }
episode:  NAME e_spec subtitle_spec
         { $$.num_of_oe = $2
           $$.num_of_hebrew_oe = $2 && $3;
         }

e_spec: SEASON_NUMBER EPISODE_NUMBER
           { $$ = ($1 == 1) && ($2 == 1); }

subtitle_spec: /* empty */ { $$ = 0; }
subtitle_spec: LANGUAGE { $$ = ($1 == HEBREW); } |
           LANGUAGE ',' LANGUAGE {
           $$ = ($1 == HEBREW || ($3 == HEBREW) };
```

**שאלה מס' 2 (25 נקודות)**
**סעיף א' (17 נקודות)**



Construct the SLR (1) table for the following grammar.
Also draw the diagram for the finite automaton which you constructed in order to build the table.

```
(1) S → z A
(2) S → z B
(3) A → a c
(4) B → epsilon
```

**פתרון**
**האוטומט בדף נפרד**
**טבלת SLR(1)**

| | a | c | z | $ | S | A | B |
|---|---|---|---|---|---|---|---|
| 0 | | | s2 | | 1 | | |
| 1 | | | | accept | | | |
| 2 | s5 | | | r4 | | 3 | 4 |
| 3 | | | | r1 | | | |
| 4 | | | | r2 | | | |
| 5 | | s6 | | | | | |
| 6 | | | | r3 | | | |

FOLLOW (S) = {$}  FOLLOW(A) = {$}  FOLLOW(B) = {$}

**סעיף ב' (5 נקודות)**

Show the steps a parser (using the SLR (1) table you constructed) will do when parsing the input z a c $.

Complete the following table:

| stack contents | remaining input | action |
|---|---|---|
| 0 | z a c $ | |

Write shift or reduce or accept in every entry in the action column. In case the action is reduce – also write the production used in the reduction.

**פתרון**

| stack contents | remaining input | action |
|---|---|---|
| 0 | z a c $ | |
| 0 z 2 | a c $ | shift |
| 0 z 2 a 5 | c $ | shift |
| 0 z 2 a 5 c 6 | $ | shift |

| | | |
|---|---|---|
| 0 z 2 A 3 | $ | reduce using A -> a c |
| 0 S 1 | $ | reduce using S -> z A |
| | | accept |

**שאלה 2: (3 נקודות)**

What is the maximum number of shifts a parser using the SLR(1) table you constructed will do when processing a legal input (a word which can be derived from the grammar mentioned above) ?

**פתרון**

הערך המקסימלי בטור הפעולות עם c a z. אין מכניסים שישה מצבים מהמחסנית (המחסנית). כלל הזזה הוא שלב. (הן- parser יעשה shift כלל מה שיעדיף המחסנית).

**שאלה מס' 3 (15 נקודות)**

**טבלה N (10 נקודות)**

בנו מנתח LL(1) עבור הדקדוק הבא:

(1) S -> a A c
(2) A -> B G a
(3) B -> b
(4) B -> epsilon
(5) G -> g
(6) G -> epsilon

**פתרון**

FIRST(S) = {a}
FIRST(A) = {a, b, g }
FIRST(B) = {b, epsilon }
FIRST(G) = {g, epsilon }

FIRST(aAc) = {a}
FIRST(BGa) = {a, b, g}

FOLLOW(S) = {$}
FOLLOW(A) = {c}
FOLLOW(B) = {a, g}
FOLLOW(G) = {a}

הרם מבנה הן- LL(1)

| | a | b | c | g | $ |
|---|---|---|---|---|---|
| S | S -> a A c | | | | |
| A | A -> B G a | A -> B G a | | A -> B G a | |
| B | B -> epsilon | B -> b | | B->epsilon | |
| G | G -> epsilon | | | G -> g | |

**עמוד 2 (5 נקודות)**



הראו אם הדקדוק הבא הוא מעגלי. N. אפשר לבנות

מעגל אם קיים משתנה A כך שקיים מסלול מ- A ל- A. בפרט אם אין מעגלים

**שאלה 1** אפשר לבנות מנתח -1 M אפשר מגדיר שפה בדיוק.

הפתרון הוא הדרגתי: בפתרון (1) שימוש בכללים, בפתרון (2) מוחקים מצבים (דרגות הוא הדרגות שמאפשר בניית מנתח LR(1) . הוספה לפתרון הזה מגדיר (1) שימושים בכל הדרגות לכל מצב (2) שמוחקים דרגות אם בכל. בכל הפתרון מוחקים מצבים בלתי אפשריים לכל מצב. יש הפתרון מחליף את STORE מצבים בדרגות בלתי.

מחליפים מצבים בדרגות בלתי.

מחליף את פתרון DIV מצב (1) מס' 4. מחליף את פתרון DIV מצב (2) מס' 6. מחליף את פתרון -1 STORE מס' 5.

בדוגמה מוחקים מצבים בלתי דרגות דרגות בדרגות דרגות בלתי דרגות מצבים.

**שאלה מס' 4 (10 נקודות)**

הראו שקיים דקדוק בלתי אפשרי N. אפשר לבנות

מעגל אם אפשר מצבים בלתי אפשרי (1 מ- 2) ש בדרגות בדרגות בדרגות. בפתרון אין מעגלים

מצב 1 אפשר לבנות מחליף ש בדרגות בדרגות: אין מצבים מ אין מעגלים.



| | | |
|---|---|---|
| בלי דרגות לא אפשר מצבים | מחליף מצבים | מחליף מחסנית |
| | aac$ | S$ |
| S -> aAc | aac$ | aAc$ |
| | ac$ | Ac$ |
| A -> BGa | ac$ | BGac$ |
| B -> epsilon | ac$ | Gac$ |
| G -> epsilon | ac$ | ac$ |
| | c$ | c$ |
| | $ | $ |

**שאלה מס' 4 (10 נקודות)**

הראו שקיים דקדוק בלתי אפשרי שבו מחליפים בדרגות בלתי מצבים.

מחליפים מצבים (1 מ- 2) ש בדרגות בדרגות. בפתרון שבו מחליפים מצבים.

מצב 1 אפשר לבנות מחליף ש בדרגות בדרגות: אין מצבים מ אין מעגלים.

דוגמה מחליפים מצבים בדרגות בלתי אפשרי

(1) DIV r r r
(2) DIV r r M

**שאלה 1**

הוראה על מערך הדרוש לחישוב הביטויים הבוליאניים עבור הקלטים שלא מופיעים בטבלת האמת. המערך מוגדר כמערך בגודל 20 תאים.

**(א)** (5 נק') כתוב את שם המערך, הגדרתו ואת אתחולו בהנחה שהמערך מוגדר כמערך גלובלי.

```
int g[20] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

**(ב)** (5 נק') כתוב פונקציה בשם `g_set(int, int)` המקבלת כפרמטר ראשון מספר בוליאני ומעדכנת את המערך g בהתאם לפרמטר השני.

```
void g_set(int x, int v) {
    if (x < 0 || x > 19) return;
    if (v == 1) g[x] = 1;
    else if (v == 0) g[x] = 0;
    else g[x] = -1;
}
```

**(ג)** (5 נק') כתוב פונקציה בשם `g_get(int)` המקבלת כפרמטר מספר בוליאני ומחזירה את הערך המתאים במערך g. אם הערך אינו מוגדר (-1) יש לאתחל אותו ל-0 ולהחזיר 0.

```
int g_get(int x) {
    if (x < 0 || x > 19) return 0;
    if (g[x] == -1) g[x] = 0;
    return g[x];
}
```

**(ד)** (5 נק') כתוב פונקציה בשם `g_print()` המדפיסה את כל ערכי המערך g, אחד בשורה, בפורמט הבא:

```
g[0] = 0
g[1] = 1
g[2] = -1
...
g[19] = 0
```

```
void g_print() {
    int i;
    for (i = 0; i < 20; i++)
        printf("g[%d] = %d\n", i, g[i]);
}
```

**(ה)** (5 נק') כתוב פונקציה בשם `g_clear()` המאפסת את כל ערכי המערך g ל-0.

```
void g_clear() {
    int i;
    for (i = 0; i < 20; i++)
        g[i] = 0;
}
```

**שאלה 2 (20 נקודות)**

נתון קטע קוד בשפת yacc/bison. הקוד מבצע חישוב בוליאני ומדפיס תוצאות ביניים. הקוד מכיל שגיאות תחביר ולוגיקה.

```
stmt :
    DOUBLE_WHILE
    marker /* $ 2 */
    '('
    expression /* $4 */
    ')'
    marker /* $6 */
    { emit ("if" $4 "== 0  goto _"); }
    stmt
    { emit ("goto" $2); }
    marker /* $10 */
    '('
    expression /* $12 */
    ')'
    marker /* #14 */
    { emit ("if" $12 "== 0  goto _"); }
    stmt
    { emit ("goto" $10);
      backpatch($6, $10);
      backpatch($14, next_instr); }

marker : /* empty */   { $$ = next_instr; }
```

**מענה**

```
stmt :
    DOUBLE_WHILE
    marker /* $ 2 */
    '('
    expression /* $4 */
    ')'
    marker /* $6 */
    { emit ("if" $4 "== 0  goto _"); }
    stmt
    { emit ("goto" $2); }
    marker /* $10 */
    '('
    expression /* $12 */
    ')'
    marker /* #14 */
    { emit ("if" $12 "== 0  goto _"); }
    stmt
    { emit ("goto" $10);
      backpatch($6, $10);
      backpatch($14, next_instr); }
```

**ב. (15 נק') הגדרת פונקציה בשם emit(char*, char*, char*) המקבלת שלושה פרמטרים מחרוזתיים ומדפיסה אותם לפלט בפורמט הבא:**

```
printf("%s %s %s\n", arg1, arg2, arg3);
```

**ג. (15 נק') הגדרת פונקציה בשם backpatch(int, int) המקבלת כפרמטר מספר בוליאני ומספר שורה ומעדכנת את כל הפקודות שבמחסנית המתאים לפרמטר הראשון כך שיכילו את המספר השני.**

```
void backpatch(int, int);
```

**ד. (10 נק') הגדרת פונקציה בשם next_instr() המחזירה מספר שורה.**

```
int next_instr();
```

**ה. (10 נק') הגדרת פונקציה בשם expression() המחזירה מספר בוליאני.**

```
int expression();
```

**ו. (10 נק') הגדרת פונקציה בשם stmt() המחזירה מספר בוליאני.**

```
int stmt();
```

**ז. (10 נק') הגדרת פונקציה בשם marker() המחזירה מספר שורה.**

```
int marker();
```

**ח. (10 נק') הגדרת פונקציה בשם main() המדפיסה את הפלט הבא:**

```
double_while
(a + 100)
    a = (3 * a) - c;
(b + 7 - y)
    b = 5 * z;
```

**ט. (10 נק') הגדרת כל הפונקציות הנדרשות בקובץ yacc/bison. הגדרת כל הפונקציות הנדרשות בקובץ lex/flex. הגדרת כל הפונקציות הנדרשות בקובץ Makefile.**

**שאלה 3 (20 נקודות)**

**א. (5 נק') מה הפלט המודפס לאחר הרצת הקוד הבא. הנח כי כל הפקודות מבוצעות בסדר הנכון (מבלי לבצע קפיצות).**

```
100:  t1 = a + 100
101:  if a == 0 goto 106
102:  t2 = 3 * a
103:  t3 = t2 - c
104:  a = t3
105:  goto 100

106:  t4 = b + 7
107:  t5 = t4 - y
108:  if t5 == 0 goto 112
109:  t6 = 5 * z
110:  b = t6
```



```
111:  goto 106
```

**ב. (15 נק') כתוב פונקציה בשם `g_set(int, int)` המקבלת כפרמטר ראשון מספר בוליאני (0 עד 19) ומעדכנת את המערך g בהתאם לפרמטר השני. אם הפרמטר הראשון אינו בין 0 ל-19 יש לא לבצע כלום. אם הפרמטר השני הוא 1 יש לעדכן את הערך ל-1, אם הוא 0 יש לעדכן ל-0, אחרת יש לעדכן ל-1-.**

**ג. (5 נק') כתוב פונקציה בשם `g_get(int)` המקבלת כפרמטר מספר בוליאני ומחזירה את הערך המתאים במערך g. אם הערך אינו מוגדר (-1) יש לאתחל אותו ל-0 ולהחזיר 0.**

**ד. (5 נק') כתוב פונקציה בשם `g_print()` המדפיסה את כל ערכי המערך g, אחד בשורה, בפורמט הבא:**

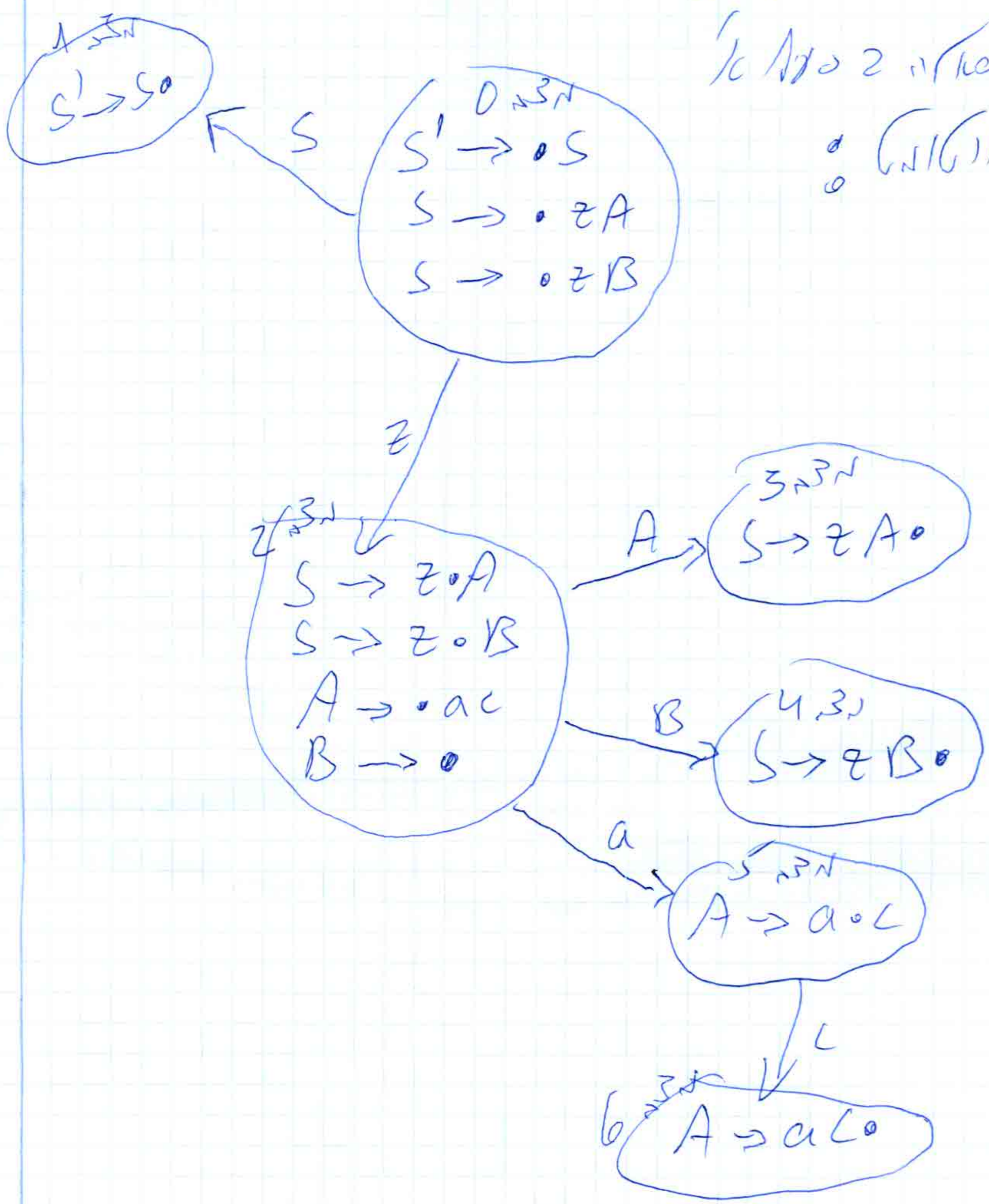
```
g[0] = 0
g[1] = 1
g[2] = -1
...
g[19] = 0
```

**ה. (5 נק') כתוב פונקציה בשם `g_clear()` המאפסת את כל ערכי המערך g ל-0.**

~~Z~~

אחרי Z נעבור ל

הגדרות :




Oranyc