פתרון מבחן סוף סמסטר – מועד א'

עבור כל שאלה במבחן להלן, תשובה א' הינה התשובה הנכונה, למעט שאלה 9 עבורה תשובה א' וגם תשובה ג' נחשבות נכונות ושאלה 11 עבורה תשובה א' וגם תשובה ד' נחשבות נכונות.

בעמוד 12 מופיעות התשובות הנכונות עבור טורים א', ב' ו-ג'.

מרצה אחראי: ד"ר איל זקס

מתרגלים: מתן פלד, אורן בניטה בן שמחון, שי גנדלמן, נדב רובינשטיין

הוראות:

א. בטופס המבחן 8 עמודים, וכן 6 דפי נוסחאות. בדקו שכל העמודים ברשותכם.
ב. משך המבחן **שלוש שעות (180 דקות).**
ג. אסור כל חומר עזר חיצוני.
ד. במבחן 20 שאלות. כל השאלות הינן חובה. כל שאלה בת 5 נקודות.
ה. קראו את כל המבחן לפני שאתם מתחילים לענות על השאלות.
ו. אין צורך להגיש את טופס מבחן זה בתום הבחינה.
ז. את התשובות לשאלות יש לסמן בטופס התשובות הנפרד בלבד.

בהצלחה!

Backpatching

נרצה להוסיף לדקדוק מבנה בקרה של לולאה עם בדיקת יציאה באמצע, במקום בהתחלה או בסוף. לדוגמא:

```
midloop
   read(&buf);
midtest (iseof())
   print(buf);
midrepeat;
```

בדוגמא הנ"ל בתחילת הלולאה נקרא מקובץ, אח"כ נבדוק אם הגענו לסוף הקלט ואם כן נצא מהלולאה, אחרת נדפיס את מה שקראנו, ואז נקפוץ לתחילת הלולאה. כמו שניתן לראות, לולאה מסוג זה שימושית מאוד עבור מקרים בהם אנו נדרשים לקרוא מידע מהקלט עד שמגיעים ל-sentinel כלשהו, כי היא חוסכת שכפול של read והבדיקה לפני תחילת הלולאה.
נוסיף לדקדוק את הכלל הבא עבור מבנה בקרה זה:

S → <u>midloop</u> S <u>midtest</u> <u>LPAREN</u> B <u>RPAREN</u> S <u>midrepeat</u> <u>;</u>

שימו לב:

א. בדקדוק אותו אנו מרחיבים יש למשתנים S ו-B כללי גזירה נוספים, ואין break.
ב. למשתנה S יש תכונת nextlist ולמשתנה B תכונות truelist ו-falselist במסגרת הטלאה לאחור של חישוב מקוצר - short-circuit evaluation, backpatching.
ג. ניתן להניח שהקוד של משתנה S תמיד מסתיים ב-goto, כולל עבור משתנים S הנגזרים לפעולות read ו-print וכן midloop כבדוגמא לעיל.
ד. אין לשנות את הדקדוק, למעט הוספת מרקרים M, N שנלמדו בכיתה בלבד.
ה. אין להשתמש בכללים סמנטיים באמצע כלל גזירה.
ו. אין להשתמש במשתנים גלובליים בזמן קומפילציה.

נמספר מופעים של S ונציין מספר מקומות [P8] – [P1] בכלל הגזירה:

S → <u>midloop</u> [P1][P2] S_0 [P3] <u>midtest</u> <u>LPAREN</u> B [P4][P5] <u>RPAREN</u> [P6][P7] S_1 [P8] <u>midrepeat</u> <u>;</u>

בבואנו להוסיף כלל סמנטי בכדי לייצר קוד תוך שימוש בחישוב מקוצר והטלאה לאחור:

1. היכן יש למקם מרקרי M?
   - א. שלושה מרקרים: אחד לפני S_0, אחד אחרי S_0, ואחד לפני S_1, כלומר ב-[P2], [P3], ו-[P7]
   - ב. לפני S_0, כלומר ב-[P2]
   - ג. אחרי S_0, כלומר ב-[P3]
   - ד. אחרי B, כלומר ב-[P4]

2. היכן יש למקם מרקרי N (מרקר N ימוקם לפני מרקר M באותה נקודה)?
   - א. ניתן לייצר קוד ללא שימוש במרקרי N כלל
   - ב. לפני S_0, כלומר ב-[P1]
   - ג. אחרי B, כלומר ב-[P4]
   - ד. שני מרקרים: אחד לפני כל S, כלומר ב-[P1] וב-[P6]

3. מה יכיל ה-nextlist של משתנה הכלל של מבנה הלולאה, כלומר של ה-S בצד שמאל של כלל הגזירה?
   א. אף תשובה אינה נכונה
   ב. את ה-nextlist של S_1
   ג. את ה-nextlist של S_0
   ד. איחוד של ה-nextlist-ים של S_0 ושל S_1

4. היכן ימוקם הקוד לו נעשה emit בתוך הפעולה הסמנטית של הכלל?
   א. ניתן לייצר קוד ללא כל emit בתוך הפעולה הסמנטית של הכלל
   ב. חובה לעשות emit, הקוד ימוקם לאחר הקוד של S_1
   ג. חובה לעשות emit, הקוד ימוקם לאחר הקוד שמחשב את תוצאת הביטוי הבוליאני B
   ד. חובה לעשות emit, הקוד ימוקם לפני הקוד של S_0, כלומר בהתחלה

5. לאילו כללים עלינו להוסיף תכונות סמנטיות, בנוסף לאלו שראינו בכיתה ומפורטים לעיל?
   א. אין צורך להוסיף תכונות סמנטיות
   ב. לכלל S
   ג. לכלל B
   ד. לכללים S ו-B

<u>פתרון שאלות 1-5</u>: היות ונתון שכל S מסתיים ב-goto, די לבצע backpatching ל-S_1.nextlist ואין צורך להוסיף N ב-P8 (או כל N שהוא) או לבצע emit "goto M_1.quad" (או כל emit שהוא)

S → <u>midloop</u> M_1 S_0 M_2 <u>midtest</u> <u>LPAREN</u> B <u>RPAREN</u> M_3 S_1 <u>midrepeat</u> <u>;</u>
{ backpatch(S_0.nextlist, M_2.quad);
backpatch(S_1.nextlist, M_1.quad);
backpatch(B.falselist, M_3.quad);
S.nextlist = B.truelist; }

אופטימיזציה

קומפיילרים רבים מממשים אופטימיזציה שמגוללת לולאות במלואן (complete loop unrolling) ומחליפה לולאה בשכפול של הפעולות בתוכה כמספר האיטרציות של הלולאה. נניח כי בשפה אין הפעולות break או continue.

לדוגמא קטע הקוד הבא:

```
int i = 0;
do {
   sum = sum + 5;
   i = i + 1;
} while (i < 4);
```

יוחלף ע"י הקוד הבא:

```
int i = 0;

sum = sum + 5;
i = i + 1;
sum = sum + 5;
i = i + 1;
sum = sum + 5;
i = i + 1;
sum = sum + 5;
i = i + 1;
```

6. מתי **לא** ניתן לבצע את האופטימיזציה complete loop unrolling?
   א. כאשר מספר האיטרציות של הלולאה איננו ניתן לחישוב בזמן קומפילציה.
   ב. כאשר ישנן בלולאה פעולות המקצות זכרון באופן דינאמי.
   ג. כאשר ישנן בלולאה פעולות printf.
   ד. טענות (1) ו-(2) נכונות.
   ה. טענות (2) ו-(3) נכונות

פתרון: בכדי לשכפל לולאה במלואה חובה לדעת כמה איטרציות יש ללולאה; אין מניעה לשכפל פעולות הדפסה או פעולות הקצאת זכרון דינאמי המופיעות בתוך הלולאה.

.

במהלך התרגול, שלושה סטודנטים התווכחו בינהם:

- אופיר טוען שישנם מצבים בהם לקומפיילר מסוג JIT יש יתרון בהפעלת אופטימיזציה מסוג complete loop unrolling
- אורנה טוענת שעדיף להריץ ניתוח חיות משתנים ואופטימיזציית dead code elimination לאחר הפעלת אופטימיזציית complete loop unrolling, מאשר לפניה
- מוחמד טוען שאת האופטימיזציה ניתן לבצע רק על קוד מכונה (בשלב ה backend של הקומפיילר)

7. מי מהם צודק?
   א. אורנה ואופיר צודקים
   ב. אורנה ומוחמד צודקים
   ג. אופיר ומוחמד צודקים

ד. רק אורנה צודקת
ה. רק אופיר צודק
ו. רק מוחמד צודק

פתרון: אופיר צודק, לקומפיילר הפועל בזמן ריצה יכולת רבה יותר לדעת כמה איטרציות יש ללולאה, לדוגמא כאשר מדובר בקבוע שערכו ידוע רק בזמן ריצה (משתנה סביבה לדוגמא), או כאשר ניתן לדעת מהו מספר האיטרציות השכיח.
אורנה צודקת: עדיף להריץ ניתוח חיות משתנים ואופטימיזציית dead code elimination אחרי הפעלת אופטימיזציית complete loop unrolling, מאשר לפניה, היות ומשתנים שהיו חיים לפני complete loop unrolling יכולים להפסיק להיות חיים ולהתבטל ע"י dead code elimination לאחר complete loop unrolling. דוגמא לכך נתונה בשאלה נפרדת אודות פעולה i=i+1 בלולאה שבה משתנה i נחשב חי ע"פ dead code elimination, וכן בדוגמא המפורשת המכילה את הפעולה x = x – 1 בלולאה אשר ניתן לבטלה ע"י dead code elimination לאחר complete loop unrolling (ולהשאר עם שתי פעולות בלבד), אך לא ניתן לבטלה ללא complete loop unrolling.
הערה: הרצת **ניתוח חיות משתנים** מאפשרת לבצע אופטימיזציה dead code elimination מסוג useless code elimination, לעומת unreachable code elimination שדי בבניית גרף זרימת הבקרה לביצועה.
הערה: אם בתוך הלולאה יש קוד המזוהה כמת לפני הפעלת complete loop unrolling, מסוג useless או מסוג unreachable, הוא יזוהה ככזה גם לאחר מכן; במקרה כזה ביטול הקוד המת מוקדם יותר יחסוך זמן קומפילציה. לחסכון בזמן הקומפילציה ישנה חשיבות, אך אופטימיזציות של קומפיילר כעקרון משקיעות זמן קומפילציה על מנת לחסוך בזמן ריצה, אשר נחשב חשוב יותר. כאמור, הפעלת complete loop unrolling יכולה להפוך קוד נוסף ל"מת" (בניגוד, לדוגמא, ל-constant folding), ובכך לחסוך זמן ריצה; על חשבון זמן קומפילציה. החיסכון בזמן קומפילציה ע"י הרצת dead code elimination בשלב מוקדם הינו משני וכללי - איננו מאפיין את אופטימיזציית complete loop unroll.
הטענה של מוחמד איננה נכונה, ניתן לבצע האופטימיזציה בשלב ה-backend אך גם בשלב ה-middle-end היות והיא מסתמכת על ייצוג הביניים בלבד, ואיננה מסתמכת על תכונות המעבד/יעד הקומפילציה.

נתון קטע קוד הביניים הבא :

```
        X = 4
        N = X + 1
        M = Y + N
Label0: Y = Y * X
        X = X - 1
        if X > 0 goto Label0
        Z = Y + 5
```

על קוד הביניים הכולל שמכיל קטע זה הופעלה תחילה אנליזת חיות משתנים, ודיווחה כי **לאחר** קטע קוד זה חיים המשתנים M ו-Z, ורק הם.

כעת עליך להריץ את האופטימיזציות השונות שנלמדו בקורס כולל אופטימיזציית complete loop unrolling עד לקבלת קוד אופטימלי.

8. איזו אופטימיזציה **אינה** מתבצעת?
   א. Common sub-expression elimination
   ב. Useless code elimination
   ג. Algebraic simplification
   ד. Constant propagation
   ה. Constant folding

פתרון: בדוגמא הנתונה אין אפשרות לבצע common sub-expression elimination, כן ניתן לבצע את האופטימיזציות הבאות:
להחליף שימוש ב-X ע"י שימוש ב-4 (constant propagation),
להחליף 4+1 ב-5 (constant folding),
לבטל הפעולה N=5 (useless code elimination) לאחר שהחלפת השימוש ב-N ע"י 5 הפכה את N ל"מת",
להחליף הפעולה Y = Y * 4 בפעולה Y = Y << 2 (algebraic simplification) לאחר הפעלת complete loop unrolling.

9. כמה בלוקים בסיסיים היו לפני הרצת האופטימיזציות, וכמה לאחר הרצת האופטימיזציות?
   א. לפני 3 ואחרי 1
   ב. לפני 2 ואחרי 1
   ג. לפני 3 ואחרי 2
   ד. לפני 4 ואחרי 2

פתרון: האופטימיזציה מבטלת לולאות ובכך מפשטת את גרף זרימת הבקרה ומקטינה את מספר הבלוקים הבסיסיים מ-3 ל-1 בדוגמא.
היות ולא צויין במפורש שאין קפיצה נוספת ל-Label0, קפיצה שתחייב שימור לייבל זה, גם תשובה ג' וגם תשובה א' נחשבו כנכונות וזיכו ב-5 נקודות.

10. על קוד הביניים הכולל שמכיל קטע זה הופעלה תחילה גם אנליזת הגדרות מגיעות (reaching definitions), ודיווחה כי לבלוק הבסיסי הראשון המתחיל בפקודה X=4 מגיעה רק הגדרה יחידה של Y=3 של Y.
כעת הריצו שוב את האופטימיזציות השונות שנלמדו בקורס כולל complete loop unrolling עד לקבלת קוד אופטימלי. מהו מספר פקודות קוד הביניים המתקבל?
   א. 2
   ב. 1
   ג. 0
   ד. 3
   ה. 7

פתרון: ישארו רק שתי פקודות, המשימות ערכים לשני המשתנים החיים M ו-Z.
היות ולא ידוע דבר אודות הקוד המשתמש במשתנים חיים אלה, לא ניתן להמשיך ולאפטם אותם לדוגמא ע"י constant propagation.

Data Flow Analysis

11. תזכורת: בלוק בסיסי A **שולט על** בלוק בסיסי B, אם כל מסלול מבלוק ההתחלה ל-B חייב לעבור דרך A.
לאחר חישוב הגדרות מגיעות reaching definitions וחישוב בלוקים שולטים dominating blocks, מי מבין הטענות הבאות נכון?
   א. אף טענה איננה נכונה
   ב. אם לתחילת בלוק בסיסי B מגיעה הגדרה יחידה d עבור משתנה נתון x, אזי בלוק בסיסי Bd המכיל את d חייב לשלוט על בלוק B?
   ג. אם לתחילת בלוק בסיסי B מגיעות שתי הגדרות d1, d2 עבור אותו משתנה נתון x, אזי d1 ו-d2 יכולות להמצא באותו בלוק בסיסי Bd?
   ד. אם לתחילת בלוק בסיסי B מגיעות שתי הגדרות d1, d2 עבור אותו משתנה נתון x, כאשר d1 ו-d2 נמצאים בבלוקים שונים B1 ו-B2 בהתאמה, אזי יתכן שגם B1 שולט על B וגם B2 שולט על B?

פתרון: ב' איננו נכון - יתכן מסלול בו יופיע שימוש במשתנה x מבלי שהוגדר תחילה. הקוד יכול להיות תקין היות ומסלול כזה בגרף הבקרה יכול לא להתבצע לעולם.
ג' איננו נכון – אם d1, d2 נמצאות באותו בלוק בסיסי Bd אזי רק אחד מהם יכול להגיע לסופו של Bd, ומכאן לתחילתו של B.
ד' נכון אם B1=B או B2=B, כפי שקורה לדוגמא בלולאה; היות ולא צויין במפורש שבלוק נחשב כשולט בעצמו (כפי שנדרש על מנת ש-ד' יהיה נכון), גם תשובה א' וגם תשובה ד' נחשבו כנכונות וזיכו ב-5 נקודות.

12. בהמשך, נניח שזיהינו את כל הלולאות בתכנית; מי מבין הטענות הבאות נכון?
   א. אם הגדרה d1 שנמצאת בבלוק בסיסי B1 מגיעה לתחילת B1, אזי בלוק B1 בהכרח נמצא בלולאה?
   ב. אם אף הגדרה שנמצאת בבלוק בסיסי B1 **לא** מגיעה לתחילת B1, אזי B1 בהכרח **לא** נמצא בלולאה?
   ג. טענות (1) ו-(2) נכונות
   ד. אף טענה איננה נכונה

פתרון: א' נכון - אם הגדרה בבלוק מגיעה לתחילתו, חייב להיות מסלול מהבלוק לעצמו, כלומר הבלוק בלולאה.
ב' איננו נכון – בלולאה המתחילה בבלוק B1 ומסתיימת בבלוק B2 המגדיר את כל המשתנים המוגדרים ב-B1, אף הגדרה של B1 לא תגיע לתחילתו

13. נאמר שבלוק בסיסי A "**קודם**" לבלוק בסיסי B אם ישנה קשת מ-A ל-B (A is predecessor block of B).
עבור כל הגדרה d1 שמגיעה לתחילת בלוק בסיסי כשלהו B, נרצה לדעת **מהיכן** מגיעה d1 ל-B; כלומר, **דרך** אילו בלוקים A ה"קודמים" ל-B מגיעה d1. מי מבין הטענות הבאות נכון?
   א. טענות (3) ו-(2) נכונות
   ב. הגדרה d1 שמגיעה לתחילת B, מגיעה אליו **דרך** בלוק "קודם" A אם ורק אם הגדרה d1 מגיעה לסוף בלוק A.
   ג. ניתן לחשב ישירות את הבלוקים הקודמים דרכם מגיעה כל הגדרה, יחד עם פתרון בעיית ההגדרות המגיעות, באמצעות פתרון בעיית DFA אחת.

ד. אף טענה איננה נכונה

פתרון: ב' נכון - ע"פ ההגדרה, הגדרה מגיעה לתחילת בלוק אם היא באיחוד ההגדרות המגיעות לסוף בלוק קודם כלשהו, לכן נוכל לדעת מהיכן הגיעה ההגדרה לבלוק ע"י בדיקת ההגדרות המגיעות לסוף כל בלוק קודם.
ג' נכון – נוכל להגדיר פריטים המציינים הגדרה בצירוף בלוק קודם דרכו הגיעה ההגדרה, וכאשר הגדרה זורמת דרך בלוק היא תשנה את הבלוק הקודם המצורף.

14. נתונה פעולה d:i=i+1 אשר מגדירה ומשתמשת במשתנה i, ונמצאת בבלוק בסיסי B אשר נמצא בתוך לולאה. מי מבין הטענות הבאות נכון?

א. אם d היא ההגדרה היחידה של i בתוך הלולאה, אנליזת חיות משתנים בהכרח תדווח שהמשתנה i חי בסוף בלוק B.
ב. טענות (1) ו-(3) נכונות
ג. אם קיימת הגדרה נוספת של i בתוך הלולאה, אנליזת הגדרות מגיעות בהכרח תדווח שהגדרה d **איננה** מגיעה לתחילת בלוק B.
ד. אף טענה איננה נכונה.

פתרון: א' נכון – הגדרה יחידה בתוך לולאה בהכרח תגיע לתחילת הלולאה, ומשם לבלוק B המכיל את ההגדרה.
ג' איננו נכון – יתכן ששתי ההגדרות של i בלולאה נמצאות בשני בלוקים שונים, אחד ב-then והשני ב-else של מבנה בקרה if-then-else בתוך הלולאה. במקרה כזה שתי ההגדרות תגענה לתחילת B.

15. מי מבין הטענות הבאות נכון?

א. אם נדע שהגדרה d:i=i+1 של משתנה i, מגיעה לשימוש יחיד והוא d עצמו, ניתן לבטל את הפקודה d במסגרת הרחבה של useless code elimination.
ב. טענות (1) ו-(3) נכונות
ג. באנליזת DFA המחשבת עבור כל הגדרה d של משתנה i, אם ההגדרה d מגיעה לשימוש **יחיד** של משתנה i, ומדווחת שימוש יחיד זה, הסריקה היא **קדמית**.
ד. אף טענה איננה נכונה.

פתרון: א' נכון – פקודה d איננה חסרת שימוש שכן היא משתמשת בעצמה, אך ניתן להרחיב האופטימיזציה על מנת לזהות מקרים אלה כחסרי שימוש "פרקטי" ולבטלם. הדבר אנלוגי למציאת מעגלים ושיחרורם במסגרת garbage collection.
ג' איננו נכון – אנליזה המזהה משתנים בעלי שימוש יחיד דומה לאנליזת חיות משתנים המזהה משתנים בעלי שימוש אחד או יותר, והיא סריקה **אחורית**. עליה לציין בכל פריט בנוסף למשתנה גם את מיקום השימוש (בדומה לפריטים של הגדרות מגיעות המציינים בנוסף למשתנה גם את מיקום ההגדרה), על מנת לזהות מתי לשני בלוקים ממשיכים (successors) ישנם שני שימושים שונים לאותו משתנה, במסגרת הסריקה האחורית.

ניתוח תחבירי

בשאלות הבאות, אותיות גדולות A,B,... הם משתנים, S הוא המשתנה ההתחלתי, ואותיות קטנות a,b,... הם טרמינלים.

נתונים הדקדוקים הבאים:

G1:	G2:
$S \to AB$ $A \to aA \mid \epsilon$ $B \to ba \mid bBa$	$S \to A$ $A \to AB \mid ab$ $B \to b$

16. איזו מבין הטענות הבאות לגבי **השפות** המוגדרות ע"י דקדוקים אלה, נכונה?
   א. גם השפה המוגדרת ע"י G1 וגם השפה המוגדרת ע"י G2 נמצאות ב-LL(1)
   ב. רק השפה המוגדרת ע"י G1 נמצאת ב-LL(1)
   ג. רק השפה המוגדרת ע"י G2 נמצאת ב-LL(1)
   ד. אף אחת מהשפות אינה ב-LL(1)

פתרון: השפה המוגדרת ע"י G1 היא $\{a^*b^na^n \mid n>=1\}$ והיא ב-LL(1), הדקדוק עצמו כמעט מוכיח זאת, יש להחליף את כללי הגזירה של B בכללים: B→bC C→a|bCa המתקבלים מביצוע left-factoring והצבה.
השפה המוגדרת ע"י G2 היא (ab)*b כלומר רגולרית, ולכן ב-LL(1).

נתונים הדקדוקים הבאים:

G3:	G4:	G5:
$S \to aAb \mid aBc$	$S \to aAb \mid aBc$	$S \to AB \mid A$
$A \to a$	$A \to ac$	$A \to cAa \mid b$
$B \to a$	$B \to ab$	$B \to b$

17. אילו **מהדקדוקים** הנ"ל נמצאים ב-$LR(0)$?
   א. G4
   ב. G3, G4
   ג. G3, G4, G5
   ד. G3

פתרון: G3 איננו דקדוק LR(0) בגלל קונפליקט reduce/reduce של כללי A ו-B
G4 הוא דקדוק ב-LR(0), בניית הטבלה נטולת קונפליקטים
G5 איננו דקדוק LR(0) בגלל קונפליקט shift/reduce של כלל B→b וכלל S→A

18. אילו **מהדקדוקים** הנ"ל נמצאים ב-$LR(1)$?
   א. G3, G4, G5
   ב. G4, G5
   ג. G3, G5
   ד. G3, G4

פתרון: ברור ש-G4 גם ב-LR(1)
קל לראות שהקונפליקט של G3 ב-LR(0) נפתר ב-LR(1)
בניית הטבלה מראה שגם הדקדוק G5 ב-LR(1)

נתונות פעולות אריתמטיות בינאריות - @,#.

- בעל אסוציאטיביות שמאלית, ו-@ בעל אסוציאטיביות ימנית. כמוכן @ בעל קדימות גבוהה יותר משל #.

נתון הביטוי:

$$1 \ @ \ 2 \ @ \ 3 \ \# \ 4 \ @ \ 5 \ \# \ 6 \ \# \ 7$$

19. אילו מהסוגריים הבאות מתאימות לסדר הפעולות בו יש לחשב את הביטוי הנתון?
   א. (((1 @ (2 @ (3 # ((4 @ (5 # (6 # 7
   ב. 1 @ (2 @ ((3 # 4) @ ((5 # 6) # (7)))
   ג. ((1 @ 2) @ (3 # ((4 @ (5 # (6 # 7))
   ד. ((1 @ 2) @ ((3 # 4) @ (5 # (6 # 7))

פתרון: קודם יש לבצע @ (בין 1,2,3 ובין 4,5) ורק אח"כ יש לבצע #. כאשר יש לבצע מספר פעולות דומות ברצף, יש לבצען מימין לשמאל עבור @, ומשמאל לימין עבור #.

נתון קוד bison הבא:

```
%{
#include <stdio.h>
int yylex();
void yyerror(char const *);
%}

%%

S:  A           {printf("1");}
    | B         {printf("2");}
;
A:  'a'         {printf("3");}
    | 'a' M A   {printf("4");}
;
B:  'b'         {printf("5");}
    | B M 'b'   {printf("6");}
;
M:  %empty      {printf("$");}
;
%%

int yylex() {return getchar();}
void yyerror(char const * s) {printf("ERR\n"); exit(1);}
int main() {return yyparse();}
```

כאשר הסימון %empty מסמן גזירה של המילה הריקה, והפונקציה yylex() מחזירה את התו הבא בקלט.

20. מה יהיה פלט המנתח המתקבל בריצה על הקלטים "aaaaa" ו-"bbbbb" בהתאמה?

א. 5$6$6$6$62 ,$$$$344441
ב. $$$$566662 ,$$$$344441
ג. $$$$566662 ,3$4$4$4$41
ד. 5$6$6$6$62 ,3$4$4$4$41

פתרון: ע"י לציור עצי הגזירה של aaaaa ו-bbbbb, כולל כללי אפסילון, ניתן לראות כיצד יבנו העצים ע"י פעולות reduce בסריקה מלמטה למעלה.

טור א'

שאלה	שאלה מס'	תשובה נכונה
1	5	ד
2	2	א
3	1	א
4	3	ב
5	4	ב
6	6	ג
7	7	ו
8	8	א
9	9	א
10	10	ה
11	11	ג+ב
12	12	ד
13	13	ב
14	14	א
15	15	ג
16	16	ג
17	17	א
18	18	א
19	19	ג
20	20	ב

טור ב'

שאלה	שאלה מס'	תשובה נכונה
1	3	ד
2	2	ד
3	5	ד
4	4	ג
5	1	ג
6	6	ג
7	7	ד
8	8	ב
9	9	א
10	10	א
11	11	ג+ב
12	12	א
13	15	ג
14	13	ג
15	14	ג
16	16	ג
17	17	א
18	18	ד
19	20	ג
20	19	ג

טור ג'

שאלה	שאלה מס'	תשובה נכונה
1	1	א
2	5	ד
3	2	א
4	4	ג
5	3	ד
6	6	ג
7	7	ג
8	8	ג
9	9	ב
10	10	ד
11	11	ד+א
12	12	ב
13	15	ב
14	14	ב
15	13	ג
16	16	ג
17	17	ב
18	18	ג
19	19	ד
20	20	ב