שאלון למטלת מנחה (ממ״ן)16

שם הקורס: קומפילציה	**מס׳ הקורס: 20364**
שם המטלה: ממ״ן 16	**מס׳ המטלה: 16**
משקל המטלה: 15 נקודות	**מחזור: א 2025**

מספר השאלות: 1

מועד משלוח המטלה: 21.3.2025

שאלה 1 (100%)

1. פרוייקט המהדר

בפרוייקט זה עליכם לתכנן ולממש חלק קדמי של מהדר, המתרגם תוכניות משפת המקור CPL לשפה Quad.

שפת המקור CPL (Compiler Project Language) היא שפה דמוית פסקל או C, אך מוגבלת מהן בהרבה.

שפת הביניים Quad היא שפת פשוטה. השפות תוגדרנה בסוף המטלה.

2. תיאור פעולת המהדר

2.1. מה עושה המהדר?

המהדר יבצע את כל שלבי ההידור (החלק הקדמי) כפי שנלמדו בקורס, החל בניתוח לקסיקלי, דרך ניתוח תחבירי ובדיקות סמנטיות, ועד לייצור קוד ביניים בשפת Quad.

המהדר יקבל קובץ קלט המכיל תוכנית בשפת CPL. כפלט, ייצר המהדר קובץ המכיל תוכנית בשפת Quad. תוכלו להריץ את תוכניות ה-Quad הנוצרות בעזרת מפרש (interpreter) שנמצא באתר הקורס.

2.2. הממשק

המהדר יהיה תוכנית המופעלת משורת הפקודה של Windows.

שמו של המהדר הוא cpq (קיצור של CPL to Quad).

קובץ הריצה צריך להיקרא cpq.exe.

הקובץ עם הפונקציה הראשית של המהדר (main) צריך להיקרא cpq.c.

<u>קלט</u> – המהדר מקבל כפרמטר יחיד שם של קובץ קלט (קובץ טקסט המכיל תוכנית בשפת CPL).

הסיומת של שם קובץ הקלט צריכה להיות ou.

שורת הפקודה היא: cpq <file_name>.ou

<u>פלט</u> – המהדר יוצר קובץ טקסט עם שם זהה לשם קובץ הקלט ועם סיומת qud.

קובץ זה מכיל את תוכנית ה-Quad שנוצרה.

<u>טיפול בשגיאות ממשק</u> – במקרה של שגיאה בפרמטר הקלט, בפתיחת קבצים וכדומה, יש לסיים את הביצוע בצירוף הודעת שגיאה מתאימה למסך (stderr). במקרה כזה אין לייצר קובץ פלט.

כחלק מהטיפול בשגיאות ממשק, יש לוודא שהסיומת של קובץ הקלט היא נכונה.

שורת חותמת – יש לכתוב שורת "חותמת" עם שם הסטודנט, אשר תופיע במקומות הבאים:
ב-standard error
בקובץ ה-quad – אחרי הוראת ה-HALT האחרונה, וזאת כדי לא להפריע למפרש של שפת Quad.

2.3. טיפול בשגיאות

ייתכן שתוכנית הקלט תכיל שגיאות מסוגים שונים:
שגיאות לקסיקליות
שגיאות תחביריות
שגיאות סמנטיות

שימו לב:
במקרה של קלט המכיל שגיאה (מכל סוג שהוא) אין לייצר קובץ qud (גם לא קובץ qud ריק).
לאחר זיהוי של שגיאה לקסיקלית , תחבירית או סמנטית, יש להמשיך בהידור מהנקודה שאחרי השגיאה. זאת כדי לגלות שגיאות נוספות אם ישנן.
את הודעות השגיאה יש לכתוב ל- standard error. הודעת השגיאה צריכה לכלול את מספר השורה בה נפלה השגיאה.

3. מימוש המהדר

3.1. שימוש בכלים flex ו-bison

מומלץ להשתמש בכלי תוכנה flex & bison או בכלים דומים (לדוגמא ply או sly שהם כלים בשפת python).

flex הוא כלי אשר מייצר באופן אוטומטי מנתחים לקסיקליים. bison הוא כלי לייצור אוטומטי של מנתחים תחביריים.

ניתן לכתוב את הקומפיילר באחת מהשפות C. C++, Java, Python. מי שרוצה להשתמש בשפה אחרת מתבקש לפנות למנחה.

3.2 מבנה כללי

הקומפיילר יכול לבצע בדיקות סמנטיות וליצר את קוד ה- Quad כבר במהלך הניתוח התחבירי. זה אפשרי כי שפת CPL היא שפה פשוטה.
לחילופין ניתן לארגן את פעולת הקומפיילר באופן הבא:
המנתח התחבירי יצור `Abstract Syntax Tree (AST)` שמייצג את התוכנית המקורית.
ואז ניתן לייצר את קוד ה- Quad במעבר על העץ. את הבדיקות הסמנטיות ניתן לבצע בשלבים שונים: במהלך בנית העץ, במעבר נפרד על העץ או בזמן שמייצרים את קוד הביניים.

3.3 חישוב יעדי קפיצה

בקוד ה- Quad שמייצר המהדר עשויות להופיע פקודות JUMP או JMPZ, כאשר יעד הקפיצה הוא מספר שורה. לצורך חישוב יעדי הקפיצה, ייתכן שתבחרו להשתמש בהטלאה לאחור (backpatching), או בשיטה של ייצור קוד זמני המכיל תוויות סימבוליות (מחרוזות), ומעבר נוסף על הקוד כדי להחליף את התוויות הסימבוליות במספרי שורות.
לצורך מימוש השיטה שבה תבחרו תוכלו להחליט להחזיק בזיכרון את כל הקוד המיוצר, או שתוכלו לייצר קבצים זמניים, שבהם ייכתב הקוד בשלבי הביניים של הייצור. בדרך כלל האפשרות הראשונה פשוטה יותר.

3.3. מבני נתונים

במימוש המבנים שגודלם תלוי בקלט יש להעדיף הקצאת זיכרון דינמית על-פני הקצאה סטטית שגודלה חסום ונקבע מראש.
במימוש המבנים שגודלם קבוע וידוע מראש עדיפה כמובן הקצאה סטטית. במבנים אלה יש גם להעדיף מימוש "מונחה טבלה" , שבו מאוחסן המידע ב"טבלה" נפרדת, והקוד משמש לגישה לטבלה ולקריאתה.

מימוש טבלת הסמלים צריך לאפשר חיפוש מהיר. לכן אין להסתפק במימוש ע"י חיפוש ברשימה מקושרת שכוללת את כל הסמלים.

באופן דומה, אם אתם שומרים את פקודות ה- Quad הנוצרות ברשימה מקושרת אז יש להימנע מסריקות של הרשימה כדי למצוא את סופה כי פעולה זו (שמן הסתם תבוצע פעמים רבות) עלולה להאט את הקומפיילר באופן משמעותי. במקום זה ניתן ביחד עם כל רשימה כזאת להחזיק גם מצביע לאיבר האחרון שלה.

מותר להשתמש במבני נתונים המוגדרים בספריות (סטנדרטיות או לא סטנדרטיות) . מותר להשתמש בקוד שמממש מבני נתונים שנמצא באינטרנט אבל יש לתת קרדיט למקור.

3.4. סגנון תכנות

התוכנית שתכתבו צריכה לעמוד בכל הקריטריונים הידועים של תוכנית כתובה היטב: קריאות, מודולריות, תיעוד וכו'.

4. כיצד להגיש את הפרוייקט

4.1. תיעוד

יש לכתוב תיעוד בגוף התוכנית, כמקובל. תיעוד זה נועד להקל על קוראי התוכנית.
התיעוד צריך להבהיר קטעי קוד שאינם ברורים. עדיף לא להסביר בהערה מה שקל להבין מהקוד עצמו.

בנוסף, יש לכתוב **תיעוד נלווה**: מסמך נפרד, שאותו ניתן לקרוא באופן עצמאי, ללא קריאת התוכנית עצמה.

לתיעוד הנלווה שתי מטרות עיקריות: הסברים על שיקולי המימוש, ותיאור מבנה הקוד. יש להציג דיון ענייני בשיקולי המימוש.

אין צורך להכביר מילים. די בעמוד אחד או שניים של תיעוד.

4.2. מה להגיש

תיקיה src עם הקבצים שכתבתם.
קובץ הרצה
קובץ README עם הוראות לבנית קובץ ההרצה. אפשר להגיש גם makefile
תיעוד

4.3. בדיקת התכנית לפני ההגשה

מומלץ להשתמש במפרש של שפת Quad שנמצא באתר הקורס. בעזרתו תוכלו להריץ את תוכניות ה-Quad שיצרתם וכך לבדוק את תקינות הקוד המיוצר. כמו כן, תוכלו להיעזר בו כדי להבין את שפת Quad – תוכלו לכתוב תוכניות דוגמה קטנות בשפת Quad, ולהריץ אותן במפרש.

בנוסף לקלטים תקינים, נסו להריץ את הקומפיילר שלכם על תכניות קלט עם שגיאות (לקסיקליות, תחביריות וסמנטיות), כולל תוכניות המכילות יותר משגיאה אחת.

שפת המקור – שפת התכנות CPL
(Compiler Project Language)

1. מבנה לקסיקלי

בשפה CPL מוגדרים האסימונים הבאים:

אסימונים המייצגים מילים שמורות:

```
break case default else float if input int output switch while
```

אסימונים המייצגים "סימבולים":

```
(   )   {   }
,   :   ; =
```

אסימונים המייצגים אופרטורים:

```
RELOP:          ==  |  != | <  |  >  | >= | <=
ADDOP:          +   |  -
MULOP:          *   |  /
OR:             ||
AND:            &&
NOT:            !
CAST:         cast<int>    cast<float>
```

אסימונים נוספים:

```
ID:             letter (letter|digit)*
NUM:            digit+  | digit+.digit*

Where:  (Note: digit and letter are not tokens)
   digit:  0  |  1  | ... |  9
   letter: a  |  b  | ... |  z  |  A  |  B  | ... | Z
```

<u>הבהרות:</u>

1. בין האסימונים יכולים להופיע תווי רווח (space), תווי טאב (\t) או תווים המסמנים שורה חדשה (\n).
   תווים כאלה חייבים להופיע כאשר הם נחוצים לצורך הפרדה בין אסימונים (למשל, בין מלה שמורה לבין מזהה). בשאר המקרים, האסימונים יכולים להיות צמודים זה לזה, ללא רווח.
2. הערות בתוכנית מופיעות בין הגבולות /* .... */ (כמו בשפת C). אין קינון של הערות.
3. השפה היא case sensitive.

2. מבנה תחבירי:

Grammar for the programming language CPL

```
program -> declarations stmt_block

declarations -> declarations declaration
              | epsilon

declaration -> idlist ':' type ';'

type -> INT
        | FLOAT

idlist -> idlist ',' ID
          | ID

stmt -> assignment_stmt
      | input_stmt
      | output_stmt
      | if_stmt
      | while_stmt
      | switch_stmt
      | break_stmt
      | stmt_block


assignment_stmt -> ID '=' expression ';'

input_stmt -> INPUT '(' ID ')' ';'
output_stmt -> OUTPUT '(' expression ')' ';'

if_stmt -> IF '(' boolexpr ')' stmt ELSE stmt

while_stmt -> WHILE '(' boolexpr ')' stmt

switch_stmt -> SWITCH '(' expression ')' '{' caselist
                           DEFAULT ':' stmtlist '}'
caselist -> caselist CASE NUM ':' stmtlist
          | epsilon

break_stmt -> BREAK ';'

stmt_block -> '{' stmtlist '}'

stmtlist  ->  stmtlist stmt
               | epsilon

boolexpr -> boolexpr OR boolterm
         |   boolterm

boolterm -> boolterm AND boolfactor
         |   boolfactor

boolfactor ->  NOT '(' boolexpr ')'
            | expression  RELOP  expression

expression -> expression ADDOP  term
             | term
```

```
term -> term MULOP factor
        | factor

factor -> '(' expression ')'
          | CAST '(' expression ')'
          | ID
          | NUM
```

3. סמנטיקה

קבועים מספריים שאין בהם נקודה עשרונית הם מטיפוס int. אחרת הם מטיפוס float.

כאשר לפחות אחד האופרנדים של אופרטור בינארי אריתמטי (פלוס, מינוס ...) הוא מטיפוס float אז התוצאה של הפעלת האופרטור היא מטיפוס float ואחרת (כלומר שני האופרנדים מטיפוס int) התוצאה היא מטיפוס int.

כאשר אופרטור בינארי מופעל על אופרנדים מטיפוסים שונים , האחד מטיפוס int והשני מטיפוס float אז האופרנד מטיפוס int עובר המרה לערך מטיפוס float לפני הפעלת האופרטור.

יש להגדיר כל משתנה פעם אחת.

חילוק בין שני שלמים נותן את המנה השלמה שלהם.

פעולת השמה היא חוקית כאשר שני אגפיה הם מאותו טיפוס או שהאגף השמאלי הוא float. במקרה של השמה של ערך מסוג int למשתנה מסוג float, הערך מומר ל- float.

משפט break יכול להופיע רק בתוך לולאת while או בתוך switch. המשמעות שלו כמו בשפת C. בהיעדר break בקוד עבור case (במשפט switch) אז אחרי ביצוע הקוד עבור ה- case ממשיכים ומבצעים את הקוד עבור ה- case הבא (כמו בשפת C).

הביטוי שמופיע אחרי switch חייב להיות בעל טיפוס int. כך גם כל מספר שמופיע אחרי case.

האופרטור `cast<int>` עושה casting ל- `int`. האופרטור `cast<float>` עושה casting ל- `float`.

4. תוכנית לדוגמה:

```
/* Finding minimum between two numbers */
a, b: float;

{
    input(a);
    input(b);

    if (a < b)
       output(a);
    else
       output(b);
}
```

שפת המטרה – Quad

לכל הוראה בשפת Quad יש בין אפס לבין שלושה אופרנדים. תוכנית היא סדרה של הוראות בשפה. הפורמט המחייב של תוכנית הוא:

- הוראה אחת בכל שורה – סוג ההוראה (ה- opcode) כתוב תמיד **באותיות גדולות**.
- קוד ההוראה והאופרנדים מופרדים על ידי תו רווח אחד לפחות.
- בכל תוכנית מופיעה ההוראה HALT לפחות פעם אחת, בשורה האחרונה.

ישנם שלושה סוגי אופרנדים להוראות השפה:

1. **משתנים**. שמות המשתנים יכולים להכיל **אותיות קטנות, ספרות ו/או קו תחתון _**. (השם אינו יכול להתחיל בספרה).
2. **קבועים מספריים** (מטיפוס שלם או ממשי) הגדרתם זהה להגדרתם בשפת CPL.
3. **יעדי קפיצה:** נרשמים כמספר שלם המסמן מספר סידורי של הוראה בתכנית (החל מ-1).

למשתנים ולקבועים בשפת Quad יש טיפוס - שלם או ממשי.
אין הכרזות של משתנים. השימוש הראשון במשתנה קובע את הטיפוס שלו.
למשל `IASN foo 3` קובע שהטיפוס של foo הוא שלם. `RASN foo 7.5` קובע שהטיפוס שלו הוא ממשי. טיפוס של משתנה איננו יכול להתחלף במהלך התוכנית. ישנן הוראות שונות עבור שלמים ועבור ממשיים. אין לערבב בין הטיפוסים. קיימות גם שתי הוראות המאפשרות מעבר בין שלמים וממשיים.

בשפה אין משתנים בולאניים, הוראות השוואה מחשבות מספר: 1 עבור True ו- 0 עבור False.
כמו כן קיימת הוראת קפיצה בלתי מותנית והוראת קפיצה מותנית. (המבצעת למעשה הוראת "if not … goto …").

הוראות שפת Quad

בטבלה הבאה:

A מציין משתנה שלם

B ו- C מציינים משתנים שלמים או קבועים שלמים

D מציין משתנה ממשי

E ו- F מציינים משתנים ממשיים או קבועים ממשיים.

L מציין יעד קפיצה (מספר שורה).

<u>שימו לב:</u> A, B, C, D, E, F הם סימנים מופשטים, שיכולים לציין משתנה כלשהו. המשתנים המופיעים בפועל בתוכנית צריכים להיכתב באותיות קטנות (מותרים גם ספרות וקו תחתון).

Opcode	Arguments	Description
IASN	A B	A := B
IPRT	B	Print the value of B
IINP	A	Read an integer into A
IEQL	A B C	If B=C then A:=1 else A:=0
INQL	A B C	If B<>C then A:=1 else A:=0
ILSS	A B C	If B<C then A:=1 else A:=0
IGRT	A B C	If B>C then A:=1 else A:=0
IADD	A B C	A:=B+C
ISUB	A B C	A:=B-C
IMLT	A B C	A:=B*C
IDIV	A B C	A:=B/C

RASN	D E	D := E
RPRT	E	Print the value of E
RINP	D	Read a real into D
REQL	A E F	If E=F then A:=1 else A:=0
RNQL	A E F	If E<>F then A:=1 else A:=0
RLSS	A E F	If E<F then A:=1 else A:=0
RGRT	A E F	If E>F then A:=1 else A:=0
RADD	D E F	D:=E+F
RSUB	D E F	D:=E-F
RMLT	D E F	D:=E*F
RDIV	D E F	D:=E/F

ITOR	D B	D:= real(B)
RTOI	A E	A:= integer(E)

JUMP	L	Jump to Instruction number L
JMPZ	L A	If A=0 then jump to instruction number L else continue.

HALT		Stop immediately.

דוגמא

הנה תוכנית בשפת CPL

```
/* Finding minimum between two numbers */
a, b: float;

{
    input(a);
    input(b);

    if (a < b)
       output(a);
    else
       output(b);
}
```

הנה תרגום אפשרי לשפת Quad

RINP a
RINP b
RLSS less a b
JMPZ 7 less
RPRT a
JMP 8
RPRT b
HALT

בהצלחה