אוניברסיטת
אריאל
בשומרון

פקולטה: מדעי הטבע

מחלקה: מדעי המחשב ומתמטיקה

שם הקורס: שפות תכנות

קוד הקורס: 2-7036010-1/2

תאריך בחינה: 12/06/2013 סמ' ב' מועד א'

משך הבחינה: 3 שעות

שם המרצה: ערן עמרי

חומר עזר: דף A4 אחד (ניתן לכתוב משני צדיו)

שימוש במחשבון: לא

הוראות כלליות:

- כתבו את תשובותיכם בכתב קריא ומרווח.
- בכל שאלה או סעיף, ניתן לכתוב – **לא יודע/ת** (מבלי להוסיף דבר מעבר לכך) – ולקבל 20% מהניקוד על השאלה או הסעיף.
- אפשר להסתמך על סעיפים קודמים גם אם לא עניתם עליהם.
- יש לענות על כל השאלות.
- ניתן להשיג עד 104 נקודות במבחן.

שאלה 1 — BNF — (27 נקודות):

נתון הדקדוק (BNF) הבא:

```
<ME> ::= <num>
       | <ME> * <ME>
       | <ME> / <ME>
```

כאשר `<num>` **מתאר ערך מספרי כלשהו על-פי הגדרת** RACKET.

סעיף א' (4 נקודות):

מהי השפה שמגדיר הדקדוק? ציירו עץ גזירה עבור מילה שבה מופיעים לפחות ארבעה מספרים (כל מספר כזה צריך להיות בן שתי ספרות – שהן צמד ספרות מתוך מספר ת.ז. של כותב המבחן).

סעיף ב' (10 נקודות):

הדקדוק הנתון אינו חד-משמעי (כלומר, הוא סובל מ-ambiguity**). הראו זאת. הסבירו מדוע זו בעיה בהקשר של שפות תכנות (השתמשו בדקדוק הנתון, כדי להדגים את הבעיה והסבירו מתי היא תתעורר).**

סעיף ג' (7 נקודות):

בסעיף זה תהפכו את הדקדוק לחד-משמעי מבלי לשנות את השפה. על הדקדוק החדש שתכתבו לקיים בפרט:

- "4 * 3 / 12" **– היא מילה קבילה בשפה, וגם**
- **אם נעריך את** "4 * 3 / 12" **כך שנפרש את סימן * כפעולת כפל ואת סימן / כפעולת חילוק, אזי הערך שיתקבל יהיה:** 16.

סעיף ד' (6 נקודות):

הסבירו כיצד נפתרה הבעיה שהדגמתם בסעיף ב (כתבו הסבר קצר במשפט או שניים – הסבר ארוך לא יתקבל).

שאלה 2 — FLANG — (26 נקודות):

לצורך פתרון שאלה זו מצורף קוד ה- interpreter של FLANG (במודל ה-substitution) בסוף טופס המבחן.

נתון הקוד הבא (עדכנו אותו לפי מספר ת.ז. שלכם):

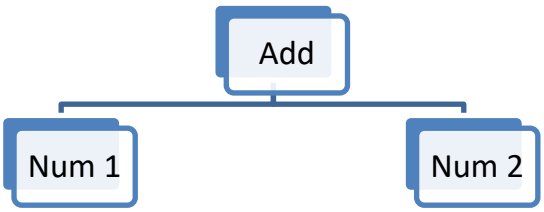
```
(run "{with {Mul-x {fun {x} {fun {y} {* x y}}}}
        {with {y <הספרה השנייה בת.ז. שלך>}
          {with {x <הספרה השלישית בת.ז. שלך>}
            {call {call Mul-x x} y}}}}")
```

הסבר: אם למשל מספר ת.ז. של כותב המבחן הוא: 012345678, אזי במקום `<הספרה השנייה בת.ז. שלך>` **יש לכתוב** 1 **ובמקום** `<הספרה השלישית בת.ז. שלך>` **יש לכתוב** 2.

סעיף א' (8 נקודות):

ציירו את עץ התחביר האבסטרקטי המתאר את הביטוי הנתון במרכאות (כלומר את התוצאה של הפעלת parse **על ביטוי זה).**

דוגמא: העץ המתאר את הביטוי `{+ 1 2}` **הוא:**



סעיף ב' (15 נקודות):

בתהליך ההערכה של ביטוי זה (הפונקציה eval) תתבצענה חמש פעולות החלפה (הפונקציה subst) – בפעולה כזו מוחלף קדקוד בעץ (שנוצר עם בנאי Id) בעץ אחר. ציירו את העץ המתקבל לאחר כל פעולת החלפה כזו (סה"כ ציירו חמישה עצים בסעיף זה לפי סדר הופעתם בחישוב).

סעיף ג' (3 נקודות):

מהי תוצאת החישוב של הביטוי כולו?

אוניברסיטת
אריאל
בשומרון

שאלה 3 — הרחבת השפה — (26 נקודות):

לצורך פתרון שאלה זו שוב נעזר בקוד ה- interpreter של FLANG המופיע בסוף טופס המבחן.

נרצה להרחיב את השפה ולאפשר מציאת מקסימום בין שני ערכים מספריים. להלן דוגמאות לטסטים שאמורים לעבוד:

```
(test (run "{maximum 7 9}")
      => 9)

(test (run "{maximum {/ 3 7} {/ 4 9}}")
      => 4/9)

(test (run "{maximum {- 3 7} {- 4 9}}")
      => -4)
```

לצורך כך נרחיב את הדקדוק באופן הבא:

```
#|
  The grammar:
    <FLANG> ::= <num>
              | { + <FLANG> <FLANG> }
              | { - <FLANG> <FLANG> }
              | { * <FLANG> <FLANG> }
              | { / <FLANG> <FLANG> }
              | { with { <id> <FLANG> } <FLANG> }
              | <id>
              | { fun { <id> } <FLANG> }
              | { call <FLANG> <FLANG> }
              | { maximum <FLANG> <FLANG> } ; Added
|#
```

סעיף א' (3 נקודות):

הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים) ל –

```
(define-type FLANG
... ראו קוד ה- interpreter מטה      ...
    [—«fill-in»— ])
```

סעיף ב' (3 נקודות):

הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים) ל –

```
(: parse-sexpr : Sexpr -> FLANG)
  ;; to convert s-expressions into FLANGs
  (define (parse-sexpr sexpr)
    (match sexpr
... מטה interpreter -ה קוד ראו      ...

      [—«fill-in»—]
      [else (error 'parse-sexpr "bad syntax in ~s" sexpr)]))
```

סעיף ג' (8 נקודות):

הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים) ל –

```
(: subst : FLANG Symbol FLANG -> FLANG)
  ;; substitutes the second argument with the third argument in the
  ;; first argument, as per the rules of substitution; the resulting
  ;; expression contains no free instances of the second argument
  (define (subst expr from to)
    (cases expr
... מטה interpreter -ה קוד ראו            ...

      [—«fill-in»—]))
```

סעיף ד' (12 נקודות):

הוסיפו את הקוד הנדרש (בתוך הסוגריים המרובעים) ל –

```
(: eval : FLANG -> FLANG)
  ;; evaluates FLANG expressions by reducing them to *expressions*
  (define (eval expr)
    (cases expr
... מטה interpreter -ה קוד ראו            ...

      [—«fill-in»—]))
```

הדרכה: תוכלו להשתמש בפונקציה max של RACKET המקבלת מספר כלשהו של Number ומחזירה את הגדול ביותר.

דוגמאות:

```
> (max 3 7 9 4)
9
> (max 3/7 4/9)

4/9
```

אוניברסיטת
אריאל
בשומרון

שאלה 4 — With vs. Call — (25 נקודות):

נתון הקוד הבא:

```
(run "{with {x {+ 2 4}}
        {with {y {* 3 5}}
            {- y x}}}")
```

סעיף א' (10 נקודות):

החליפו את הקוד הנ"ל בקוד שקול בו לא מופיעה המילה with – לצורך כך השתמשו במילה call.

סעיף ב' (15 נקודות):

הכלילו את הרעיון כדי להחליף את הקוד המודגש – בשורת קוד שאינה מכילה הפעלה של subst ומכילה הפעלה יחידה של eval (במקום שתי הפעלות):

```
(: eval : FLANG -> FLANG)
  ;; evaluates FLANG expressions by reducing them to *expressions*
  (define (eval expr)
    (cases expr
      [(Num n) expr]
      [(Add l r) (arith-op + (eval l) (eval r))]
      [(Sub l r) (arith-op - (eval l) (eval r))]
      [(Mul l r) (arith-op * (eval l) (eval r))]
      [(Div l r) (arith-op / (eval l) (eval r))]
      [(With bound-id named-expr bound-body)
       (eval (subst bound-body
                    bound-id
                    (eval named-expr)))]
      [(Id name) (error 'eval "free identifier: ~s" name)]
      [(Fun bound-id bound-body) expr]
      [(Call fun-expr arg-expr)
       (let ([fval (eval fun-expr)])
         (cases fval
           [(Fun bound-id bound-body)
            (eval (subst bound-body
                         bound-id
                         (eval arg-expr)))]
           [else (error 'eval "`call' expects a function, got: ~s"
                                                          fval)]))]))
```

הדרכה: בנו מחדש את ה-FLANG שנבנה עם With בעזרת בנאי Call.

אוניברסיטת
אריאל
בשומרון

```
---<<<FLANG>>>-------------------------------------------------------

  ;; The Flang interpreter

  #lang pl

  #|
  The grammar:
    <FLANG> ::= <num>
              | { + <FLANG> <FLANG> }
              | { - <FLANG> <FLANG> }
              | { * <FLANG> <FLANG> }
              | { / <FLANG> <FLANG> }
              | { with { <id> <FLANG> } <FLANG> }
              | <id>
              | { fun { <id> } <FLANG> }
              | { call <FLANG> <FLANG> }

  Evaluation rules:

    subst:
      N[v/x]                 = N
      {+ E1 E2}[v/x]         = {+ E1[v/x] E2[v/x]}
      {- E1 E2}[v/x]         = {- E1[v/x] E2[v/x]}
      {* E1 E2}[v/x]         = {* E1[v/x] E2[v/x]}
      {/ E1 E2}[v/x]         = {/ E1[v/x] E2[v/x]}
      y[v/x]                 = y
      x[v/x]                 = v
      {with {y E1} E2}[v/x] = {with {y E1[v/x]} E2[v/x]} ; if y =/= x
      {with {x E1} E2}[v/x] = {with {x E1[v/x]} E2}
      {call E1 E2}[v/x]      = {call E1[v/x] E2[v/x]}
      {fun {y} E}[v/x]       = {fun {y} E[v/x]}            ; if y =/= x
      {fun {x} E}[v/x]       = {fun {x} E}

    eval:
      eval(N)             = N
      eval({+ E1 E2})     = eval(E1) + eval(E2)   \ if both E1 and E2
      eval({- E1 E2})     = eval(E1) - eval(E2)    \ evaluate to numbers
      eval({* E1 E2})     = eval(E1) * eval(E2)    / otherwise error!
      eval({/ E1 E2})     = eval(E1) / eval(E2)   /
      eval(id)            = error!
      eval({with {x E1} E2}) = eval(E2[eval(E1)/x])
      eval(FUN)           = FUN ; assuming FUN is a function expression
      eval({call E1 E2}) = eval(Ef[eval(E2)/x]) if eval(E1)={fun {x} Ef}
                          = error!                otherwise
  |#

  (define-type FLANG
    [Num  Number]
    [Add  FLANG FLANG]
    [Sub  FLANG FLANG]
    [Mul  FLANG FLANG]
```

```
  [Div  FLANG FLANG]
  [Id   Symbol]
  [With Symbol FLANG FLANG]
  [Fun  Symbol FLANG]
  [Call FLANG FLANG])

(: parse-sexpr : Sexpr -> FLANG)
;; to convert s-expressions into FLANGs
(define (parse-sexpr sexpr)
  (match sexpr
    [(number: n)     (Num n)]
    [(symbol: name) (Id name)]
    [(cons 'with more)
     (match sexpr
       [(list 'with (list (symbol: name) named) body)
        (With name (parse-sexpr named) (parse-sexpr body))]
       [else (error 'parse-sexpr "bad `with' syntax in ~s" sexpr)])]
    [(cons 'fun more)
     (match sexpr
       [(list 'fun (list (symbol: name)) body)
        (Fun name (parse-sexpr body))]
       [else (error 'parse-sexpr "bad `fun' syntax in ~s" sexpr)])]
    [(list '+ lhs rhs) (Add (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '- lhs rhs) (Sub (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '* lhs rhs) (Mul (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list '/ lhs rhs) (Div (parse-sexpr lhs) (parse-sexpr rhs))]
    [(list 'call fun arg) (Call (parse-sexpr fun) (parse-sexpr arg))]
    [else (error 'parse-sexpr "bad syntax in ~s" sexpr)]))

(: parse : String -> FLANG)
;; parses a string containing a FLANG expression to a FLANG AST
(define (parse str)
  (parse-sexpr (string->sexpr str)))

(: subst : FLANG Symbol FLANG -> FLANG)
;; substitutes the second argument with the third argument in the
;; first argument, as per the rules of substitution; the resulting
;; expression contains no free instances of the second argument
(define (subst expr from to)
  (cases expr
    [(Num n) expr]
    [(Add l r) (Add (subst l from to) (subst r from to))]
    [(Sub l r) (Sub (subst l from to) (subst r from to))]
    [(Mul l r) (Mul (subst l from to) (subst r from to))]
    [(Div l r) (Div (subst l from to) (subst r from to))]
    [(Id name) (if (eq? name from) to expr)]
    [(With bound-id named-expr bound-body)
     (With bound-id
           (subst named-expr from to)
           (if (eq? bound-id from)
             bound-body
             (subst bound-body from to)))]
    [(Call l r) (Call (subst l from to) (subst r from to))]
    [(Fun bound-id bound-body)
     (if (eq? bound-id from)
```

```
        expr
        (Fun bound-id (subst bound-body from to)))]))

(: arith-op : (Number Number -> Number) FLANG FLANG -> FLANG)
;; gets a Racket numeric binary operator, and uses it within a FLANG
;; `Num' wrapper
(define (arith-op op expr1 expr2)
  (: Num->number : FLANG -> Number)
  (define (Num->number e)
    (cases e
      [(Num n) n]
      [else (error 'arith-op "expects a number, got: ~s" e)]))
  (Num (op (Num->number expr1) (Num->number expr2))))

(: eval : FLANG -> FLANG)
;; evaluates FLANG expressions by reducing them to *expressions*
(define (eval expr)
  (cases expr
    [(Num n) expr]
    [(Add l r) (arith-op + (eval l) (eval r))]
    [(Sub l r) (arith-op - (eval l) (eval r))]
    [(Mul l r) (arith-op * (eval l) (eval r))]
    [(Div l r) (arith-op / (eval l) (eval r))]
    [(With bound-id named-expr bound-body)
     (eval (subst bound-body
                  bound-id
                  (eval named-expr)))]
    [(Id name) (error 'eval "free identifier: ~s" name)]
    [(Fun bound-id bound-body) expr]
    [(Call fun-expr arg-expr)
     (let ([fval (eval fun-expr)])
       (cases fval
         [(Fun bound-id bound-body)
          (eval (subst bound-body
                       bound-id
                       (eval arg-expr)))]
         [else (error 'eval "`call' expects a function, got: ~s"
                      fval)]))]))

(: run : String -> Number)
;; evaluate a FLANG program contained in a string
(define (run str)
  (let ([result (eval (parse str))])
    (cases result
      [(Num n) n]
      [else (error 'run
                   "evaluation returned a non-number: ~s" result)])))

;; tests
(test (run "{call {fun {x} {+ x 1}} 4}")
      => 5)
(test (run "{with {add3 {fun {x} {+ x 3}}}
               {call add3 1}}")
      => 4)
(test (run "{with {add3 {fun {x} {+ x 3}}}
```

```
                {with {add1 {fun {x} {+ x 1}}}
                  {with {x 3}
                    {call add1 {call add3 x}}}}}")
      => 7)
(test (run "{with {identity {fun {x} x}}
                {with {foo {fun {x} {+ x 1}}}
                  {call {call identity foo} 123}}}")
      => 124)
(test (run "{call {call {fun {x} {call x 1}}
                        {fun {x} {fun {y} {+ x y}}}}
                  123}")
      => 124)
```