Homework: VarLang and DefineLang

Learning Objectives:

In this homework, we will test your knowledge of VarLang, DefineLang and related concepts. This includes:

1. Understanding variable scopes.
2. Understanding free vs. bound variables.
3. Writing and understanding programs in VarLang and DefineLang.
4. Understanding operational semantic rules.
5. Understanding and extending VarLang and DefineLang implementation of variable environments.

Instructions:

1. **Total points:** 48 pt
2. **Early deadline:** Feb 18 (Wed) 11:59 pm, **Regular deadline:** Feb 20 (Fri) 11:59 pm
3. **How to submit:**
   - For questions 1–3, you can write your solutions in latex or word and then convert it to PDF; or you can submit a scanned document with legible handwritten solutions. Please provide the solutions in one PDF file. **Rename the PDF to *answers.pdf*.**
   - For question 4, please submit your solution in one zip file with all the source code files (just zip the complete project's folder). **The zip file must be renamed to *q4.zip* before submitting.**
   - Bundle up the PDF(*answers.pdf*), the zip file (*q4.zip*) into another single zip file. **Rename this zip file to *hw3.zip*.** Submit *hw3.zip* to Canvas under Assignments, Homework 3 through Gradescope.

Questions:

1. **(10 pt) [Programming, Scoping, Semantics]**
   (a) (3 pt) Write a VarLang program that evaluates to `3420`. The program must make **at least 2** "holes in the scope" using nested `let` expressions.
   (b) (4 pt) Rewrite the above VarLang program into a **single** DefineLang program using define expressions. Each `let` expression must have its own `define` expression and the program should evaluate to 3420. Note that you are allowed to define multiple variables. Make sure you check the grammar and the semantics before writing your program.

(c) (3 pt) Complete the following program such that the final program evaluates to `1`.

```
$(define a _______)
$(let ( (x (+ a 5)) (y (- _______ 10)) )
      (/ x (* y _______ _______)))
```

2. **(6 pt) [Scope]**
The following VarLang program exhibits a **hole** in the scope of one or more of its variables.

```
1 (*
2     (let ((x 3) (y 5))
3       (let ((y x))
4           (- x y)
5       )
6     )
7     (let ((x 2))
8       (+ x x)))
```

In your answer:

(a) (2 pt) Identify the impacted variables(s)

(b) (4 pt) List the line number(s) that constitute the hole in the scope for each impacted variable.

3. **(12 pt) [Free and Bound Variables]**
The following questions will each present a single VarLang program which references one or more variables. For your answer to each question, please list each variable and indicate whether that variable is **free** or **bound**.

Tip: If the same name is used to refer to multiple, distinct variables, you may add an index to the variable in your answer. For e.g., "x1", "x2", which will be interpreted as indexing the first reference to / use of that name (not the first declaration) starting from the beginning of the program.

For example:

```
(let ((x 1) (y x)) (let ((x (+ x y))) (- x z)))
```

- `x1`: free – the `x` in `"(y x)"`, NOT the `x` declared in `"(x 1)"`
- `x2`: bound – the `x` in `"(+ x y)"`
- `y`: bound – the `y` in `"(+ x y)"`
- `x3`: bound – the `x` in `"(- x z)"`
- `z`: free – the `z` in `"(- x z)"`

(a) (3 pt)

```
(let ((length 12) (width 14))
   (let ((peri (+ (* 2 length) (* 2 width))))
      peri))
```

(b) (4 pt)

```
(let ((radius 3) (area (* pi (* radius radius))))
  area)
```

(c) (5 pt)

```
(+
   (let ((numerator 9) (denominator 5))
      (let ((ratio (/ numerator denominator)))
         (* ratio c)
      )
   )
   (let ((min 32))
      min))
```

4. **(20 pt) [Interpreter Extension]**
For the next two questions, you will implement an extension of the DefineLang by altering the implementation provided with the homework. Please submit a **zipped project folder *q4.zip***, archiving all the code.

(a) (10 pt) The first modification you will make to DefineLang will be to change the semantics of the `let` expression.

Change `let` expression such that the variables declared within a given `let` expression can reference each other. In other words, the variable assignments will not all be evaluated in the enclosing context, as was done in the unmodified version of Var/DefineLang.

To be precise, your updated implementation will use the following, modified operational semantics when evaluating the let expression:

$$\frac{\begin{array}{l} n_1, \ldots n_k \in \text{Identifier} \\ e_1, \ldots e_k, e' \in \text{Exp} \\ env_{k+1} \in \text{Env} \\ env_i = \text{ExtendEnv } n_i \; v_i \; env_{i+1} \text{ for } i = k..1 \\ v_i = \text{value } e_i \; env_{i+1} \text{ for } i = k..1 \end{array}}{\text{value (LetExp } ((n_1 \; e_1) \ldots (n_k \; e_k)) \; e') \; env_{k+1} = \text{value } e' \; env_1}$$

(b) (10 pt) For the second modification, make DefineLang refuse re-declaration of global Variables through `DefineDecl` expressions. If a re-declaration is detected, then don't update the global environment (`GlobalEnv`) but instead show an error "Re-declaration of variable <variabl_name>detected. Discarding redefinition".

For example,
Before the change

```
$ (define i 1)
$ (define i 2)
$ i
2
```

After the change

```
$ (define i 1)
$ (define i 2)
Re-declaration of variable i detected. Discarding redefinition.
$ i
1
```