Project 2
COMP301 Fall 2021

Deadline: Nov 26, 2021 - 23:59 (GMT+3 : Istanbul Time)

In this project, you will work in groups of two or three. To create your group, use the Google Sheet file in the following link:

Link to Google Sheets for Choosing Group Members

Note: You need to **self-enroll** to your Project2 group on BlackBoard (please only enroll to the same group number as your group in the Sheets), please make sure that you are enrolled to Project 2 - Group #YourGroup.

This project contains a bonus component specified at the end and there are two code boilerplates provided to you: use `Project2MYLET` for the project and `Project2BONUS` for the bonus. Submit a report containing your answers to the written questions in PDF format and Racket files for the coding questions to Blackboard as a zip. Include a brief explanation of your team's workload breakdown in the pdf file. If you attempt to solve the bonus question, make sure that your zip includes both `Project2MYLET` and `Project2BONUS` folders separately. Name your submission files as
p2_ member1IDno_ member1username_ member2IDno_ member2username.zip
Example: *p2_ 0064115_ fbulgur17_ 0079023_ akutuk21.zip*.

Please use *Project 2 Discussion Forum* on Blackboard for all your questions.

The deadline for this project is Nov 26, 2021 - 23:59 (GMT+3 : Istanbul Time). **Read your task requirements carefully. Good luck!**

TABLE 1. Grade Breakdown for Project 2

Question	Grade Possible
Part A	10
Part B	10
Part C	5
Part D	70
Part E	5
Total	100
Bonus	2 pts

Problem Definition: To evaluate programs, you need to understand the expressions of the language. It is the same for computers; you saw in the lecture how you can invent a language and define it for the computer to understand and evaluate.

In this project, you will define a language named MYLET that is similar to the simple LET language covered in the class. The syntax for the MYLET language is given below.

```
Program ::= Expression
            ┌─────────────────┐
            │ a-program (exp1) │
            └─────────────────┘

Expression ::= Number
               ┌──────────────────┐
               │ const-exp (num)  │
               └──────────────────┘

Expression ::= Boolean
               ┌──────────────────┐
               │ bool-exp (bool)  │
               └──────────────────┘

Expression ::= String
               ┌──────────────┐
               │ str-exp (str) │
               └──────────────┘

Expression ::= op-exp(Expression, Expression, Expression)
               ┌──────────────────────────┐
               │ op-exp (exp1 op-exp exp2) │
               └──────────────────────────┘

Expression ::= zero?-exp (Expression)
               ┌──────────────────┐
               │ zero?-exp (exp1) │
               └──────────────────┘

Expression ::= comp-exp (Expression, Expression, Expression)
               ┌────────────────────────────┐
               │ comp-exp (exp1 op-exp exp2) │
               └────────────────────────────┘

Expression ::= my-cond Expression then Expression,
               {Expression then Expression,}*
               else Expression
               ┌─────────────────────────────────────────┐
               │ my-cond (cond1 exp1 conds exps else-exp) │
               └─────────────────────────────────────────┘

Expression ::= Indetifier
               ┌──────────────────┐
               │ var-exp (var)    │
               └──────────────────┘

Expression ::= let Indetifier = Expression in Expression
               ┌───────────────────────┐
               │ let-exp (var exp1 body) │
               └───────────────────────┘
```

FIGURE 1. Syntax for the MYLET language

Part A. This part will prepare you for the following parts of the project. (10 pts)
  (1) Write the 5 components of the language:
  (2) For each component, specify where or which racket file (if it applies) we define and handle them.

Part B. In this part, you will create an initial environment for programs to run. (10 pts)
  (1) Create an initial environment that contains 3 different variables (x, y, and z). The values of x, y and z variables should be 4, 3 and 6 respectively (x=4, y=3, and z=6).
  (2) Using the environment abbreviation shown in the lectures, write how environment changes at each variable addition.

Part C. Specify expressed and denoted values for `MYLET` language. (5 pts)

Part D. This is the main part of the project where you implement the MYLET language given in the Figure 1 by adding the missing expressions.

(1) Add *bool-exp* to the language.(10 pts) Booleans are defined as any text starting with #, e.g. #true or #false; booleans are stored with # symbols.

(2) Add *str-exp* to the language. (10 pts) Strings are defined as any text starting and ending with ', e.g. 'comp301', 'program'; strings are stored with ' symbols.
Hint: *String* is an expression that is similar to *Number*, understanding the addition and implementation of *Number* may be helpful to complete this step.

(3) Add *op-exp* to the language. (10 pts) *op-exp* takes three expressions and evaluates second expression over the first and third expressions. The second expression can be as follows: **"add", "mult", "div", "sub"**.
   - add: perform addition (exp1 "add" exp2)
   - mult: perform multiplication (exp1 "mult" exp2)
   - div: perform division (exp1 "div" exp2)
   - sub: perform subtraction (exp1 "sub" exp2)

(4) Add *comp-exp* to the language.(10 pts) Similar to the *op-exp*, *comp-exp* takes three expressions and evaluates second expression over the first and third expressions. The second expression can be as follows: **"greater", "equal", "less"**.
For example,
   comp-exp(10,'greater', 9)   **;; 10 > 9 so it returns true.**

(5) Add *my-cond* to the language. (10 pts) Unlike *cond* of the LET language, the evaluation steps are a little bit different in *my-cond*. In *cond* of the LET language, starting from the first condition it evaluates expressions in order and the true condition will be evaluated as a result. However, now you need to implement *my-cond* a little bit different:
   - starting from the first condition it evaluates expressions in order.
   - this time you return the **last true** evaluated condition as a result, instead of returning the **first** condition evaluated as true.
To better understand how my-cond should be working, there is an example provided. For example,
   (let ((x 2) (y 6) (z 7))
       (cond ((> x 4) 12)
             ((< x y) 13)
             ((< y z) 15)
             (else 11)))
   ;; returns 15 instead of 13!!
So, second and third conditions are evaluated as true but it should return the last condition evaluated as true, it returns
(< y z 15)   **;; returns 15.**

(6) Add a custom expression to the language. (10 pts) The expression can be simple, but you need to clearly explain what it does and how it works. You also need to provide the syntax of the expression.

(7) Create the following test cases.(10 pts) For the custom expression: Write test cases that controls if the expression works according to your explanation of the expression.
Note: We provided several test cases for you to try your implementation. Uncomment

corresponding test cases and run `tests.rkt` to test your implementation.

Note that the implementation of the other expressions, that are same with the LET language, are already given in the .rkt file provided. We deleted the former implementations of `if` and `diff-exp`.

Part E. *Thinking question:*
Let's consider *my-cond* operation you have implemented. It is defined so that it returns the last true evaluated value, not the first true value, unlike the *cond* operation in the LET language. Now imagine that you have two my-cond implementations given but you cannot see the details of the implementations. One of those cond implementations is designed to return the last one it finds as true but starts to search from the first conditions given. The idea is to store the previously true evaluated values in a variable (top-to-bottom). On the other hand, the other cond implementation directly returns the last correct value found, it directly searches from the last conditions given instead of starting from the first conditions (bottom-to-top), and returns the first correct value it finds. Is it possible to understand the correct one among these two implementations by just trying different test cases? Please explain your reasoning in detail about whether it is possible to understand or not and why? (5 pts)

Bonus. <u>Here</u> is an alternative datatype *ropes* that allows manipulation of sequence of characters instead of the most commonly used `strings`. You can try to implement *rope*s instead of *string*s as a bonus challenge.
Note: The bonus question is worth 2 points in your overall final grade and no partial credits will be awarded. To get full credit, please implement this problem using the second code boilerplate (`Project2BONUS`) provided and write at least 6 test cases (two for each: fetch i^{th} character, concatenate, substring) in a clear way to your `tests.rkt` for us to run. Please make sure that your test cases are clear and `tests.rkt` doesn't give any errors, otherwise you won't be able to receive any credits for this question. Add your code for the bonus problem to your submission as specified in the instructions.
Hint: Define your *rope* datatype similar to the way you did in the project, clearly define your grammar and feel free to use any helper procedures.