COMP 301 - PROJECT 2

Team members:
Muhammet Eren Ozogul
Ada Yıldız
Aynur Ceren Cepni

PART A:

(1)
1-Program Text: Represents the input to the program, the source language
2-Front End: scanner(which generates lexemes, lexical items, tokens) & parser (which generates AST, syntactic structure, grammatical structure)
3-Syntax Tree: Abstract syntax tree that a parser generates.
4-Interpreter: Evaluates each expression accordingly to the AST and gives us an expressed value
5-Answer: Final result produced by the interpreter after the evaluation of AST

(2)
1-Lexical Specification: is the specification of the process of converting a sequence of characters in the source code to a sequence of tokens. This is controlled by the lexical analyzer. Defined in lang.rkt.
2-Syntax: Syntax is the grammar of our language, it specifies the way our language gets converted to an AST. The process of converting text into an AST representation is done by the parser. Defined in lang.rkt.
3-Scanner and Parser: Scanner turns an input string into a list of tokens and the parser builds up the abstract syntax tree for the compilation process. Defined in lang.rkt.
4-Data-Types: Defined in data-structures.rkt.
5-Interpreter: Interpreter reads the expressions and evaluates them. Defined in interp.rkt

PART B:

(a)
```
(define init-env
        (lambda ()
                (extend-env 'z (num-val 4)
```

(extend-env 'y (num-val 6)
(extend-env 'x (num-val 2)
(empty-env))))))

(b) (init-env) = [z=⌈4⌉, y=⌈6⌉, x=⌈2⌉]
[]
[x=⌈2⌉]
[y=⌈6⌉,x=⌈2⌉]
[z=⌈4⌉, y=⌈6⌉, x=⌈2⌉]

(c) We could implement the environments using different representations such as data-structure representation or procedural representation. In MYLET language, data-structure representation (specifically, lists) is used.

PART C:

Expressed Value: The possible values of expressions.
Denoted Value: The values bound to variables.
ExpVal: Int + Bool + List<Int> + Pair<Int,Int>
DenVal: Int + Bool +List<Int> + Pair<Int,Int>

PART D:

Solutions are in the code.