CS 474: Object-Oriented Programming Languages and Environments

Ruby Interpreter Specification Document

Name: Angela Santhosh George
UIN: 667790862

1. Overview

This document defines the refined specification for the Plain Programming Language (PPL) interpreter implemented in Ruby. The interpreter executes simple PPL programs composed of line-based instructions that manipulate integers and linked lists. The language supports variable declarations, arithmetic, list manipulation, conditionals, and program control flow. The interpreter reads a text file (*.ppl*) containing one instruction per line and executes statements sequentially from top to bottom unless control flow instructions alter this order.

2. Input File Format

- Each statement occupies exactly one line**.**
- Lines may contain comments, starting with the hash symbol (`#`). Comments are ignored by the interpreter.
- Blank lines are allowed and ignored.
- Identifiers must begin with a letter and may contain letters, digits, or underscores.
- Instructions and arguments are case-insensitive but are converted to uppercase internally.

Example:

```
INTEGER a
ASSIGN a 5
LIST L1
MERGE a L1
PRINT L1
HLT
```

3. Program Execution Model

- The interpreter maintains an environment that stores variable bindings (identifier → value).
- Each variable may store either an integer value or a linked list (implemented via a custom linked list class).
- Execution proceeds line-by-line unless altered by an IF instruction.
- The program halts when an HLT instruction is executed or the end of file is reached.
- On runtime errors, the interpreter prints the line number and error description, then shows the final environment state.

4. Instruction Set Specification

Each instruction is described below with its syntax, semantics, and error conditions.

Instructions	Purpose	Semantics	Errors
INTEGER *var*	Declares a new integer variable initialized to 0	Creates a binding in the environment: *var* → 0	If *var* is already declared → Runtime error: Identifier '<id>' already declared.
LIST *identifier*	Declares a new variable as an empty linked list.	Creates a binding: identifier → empty linked list	If *identifier* already exists → Runtime error: Identifier '<id>' already declared.
ASSIGN *identifier value*	Assigns an integer constant to an integer variable.	identifier → integer(value)	• Missing value argument → Runtime error: Wrong argument count (expected 2, got 1) • If *identifier* not declared or not an integer → Runtime error: ASSIGN target must be integer.
CHS *int*	Changes the sign of the integer value.	*int* = - *int*	If *int* not an integer → Runtime error: CHS expects integer.
ADD *int1 int2*	Adds two integer values and stores the result in the first operand.	*int1* = *int1* + *int2*	Either operand not an integer → Runtime error: ADD expects integers.
MERGE *identifier list*	Prepends a value (integer or list) to another list.	Appends a deep copy of the value bound to identifier as the first element of the list bound to list.	• Target not a list → Runtime error: MERGE target not list. • Source identifier not found → Runtime error:

			Unknown identifier.
COPY *list1 list2*	Creates a deep copy of one list into another identifier.	*list2* → deep copy of *list1*	Source is not a list → Runtime error: COPY source not list.
HEAD *list identifier*	Extracts the first element of a list and stores it in an identifier.	*identifier* → first element of list	• Source not a list → Runtime error: HEAD source not list. • Empty list → Runtime error: HEAD from empty list.
TAIL *list1 list2*	Creates a list containing all elements except the first from list1 and stores it in list2.	*list2* → *list1*.tail (i.e., all elements except the head)	Source not a list → Runtime error: TAIL source not list.
IF *identifier line_number*	Conditional jump based on value.	If: • identifier is an integer equal to 0, or • identifier is a list that is empty, then execution jumps to the specified absolute line number (1-based). Otherwise, execution continues sequentially.	• Nonexistent identifier → Runtime error: Unknown identifier. • Invalid line number → Runtime error: Invalid jump target.
PRINT *identifier*	Displays the value of a variable on screen.	Prints: *identifier* = value	Undeclared identifier → Runtime error: Unknown identifier.
HLT	Terminates program execution immediately.	Stops execution, prints final state of all variables.	

5. Program Flow

- The interpreter reads the .ppl source file line by line.
- Comments (# ...) and blank lines are ignored.
- Each valid line is tokenized into an instruction and arguments.
- Variables are stored in a symbol table with their type (int or list) and value.
- Execution proceeds sequentially, except when an IF causes a jump or HLT halts the program.
- Each instruction performs type and argument validation before execution.
- In the event of an error, the program stops and reports the exact line, instruction, and error message.
- At termination (whether normal or due to an error), all variable states are displayed as the final output.

6. Error Handling and Diagnostics

When an error occurs:

- Execution halts immediately.
- A diagnostic message is printed including:
    - line number,
    - the faulty instruction,
    - a descriptive message.
- The current environment (variable bindings) is printed after termination.

Example:

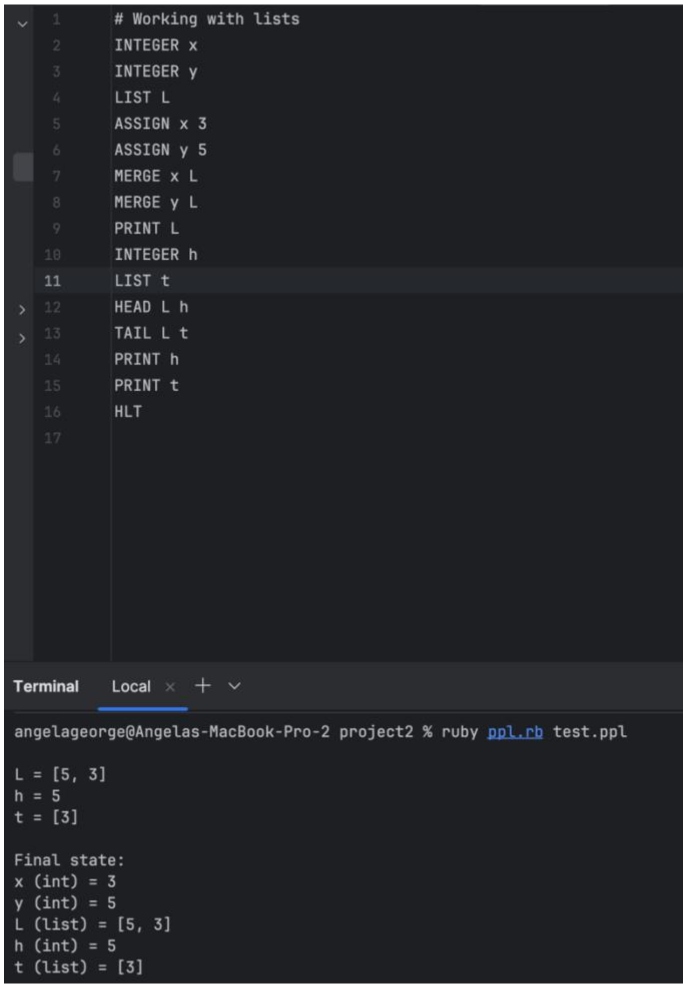
```
Runtime error at line 12:
  >> MERGE a b
  Error: MERGE target not list.

Final state:
a (int) = 5
b (int) = 0
```

7. Assumptions & Constraints

- One instruction per line; arguments are space-separated.
- Identifiers must be declared before use (INTEGER or LIST).
- ASSIGN accepts only integer constants (no variable-to-variable assignment).
- IF jumps to absolute line numbers (1-based).
- Lists and integers are distinct types with no implicit conversions.
- Execution halts immediately on encountering an error or HLT.

8. Test Cases

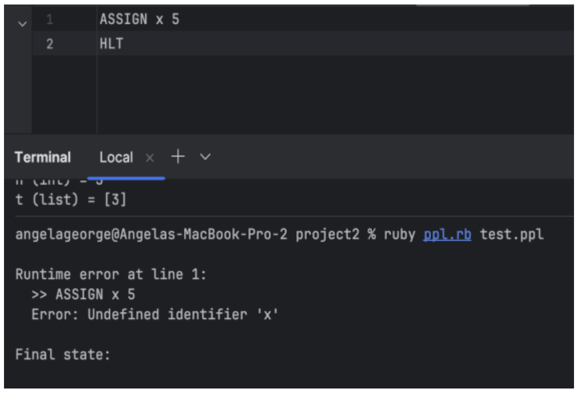
a) Valid Input Examples



```
# Working with lists
INTEGER x
INTEGER y
LIST L
ASSIGN x 3
ASSIGN y 5
MERGE x L
MERGE y L
PRINT L
INTEGER h
LIST t
HEAD L h
TAIL L t
PRINT h
PRINT t
HLT
```

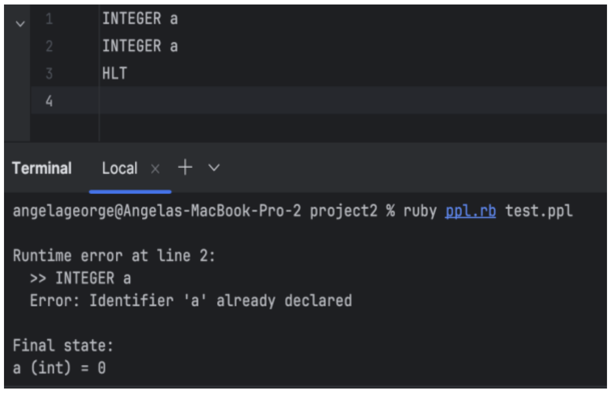
```
angelageorge@Angelas-MacBook-Pro-2 project2 % ruby ppl.rb test.ppl

L = [5, 3]
h = 5
t = [3]

Final state:
x (int) = 3
y (int) = 5
L (list) = [5, 3]
h (int) = 5
t (list) = [3]
```

b) Invalid Input Examples



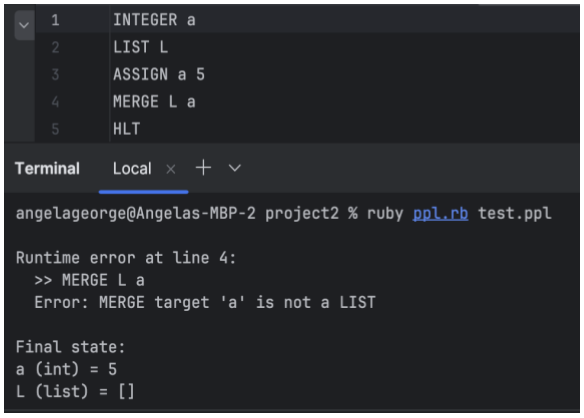
```
ASSIGN x 5
HLT
```

```
t (list) = [3]

angelageorge@Angelas-MacBook-Pro-2 project2 % ruby ppl.rb test.ppl

Runtime error at line 1:
  >> ASSIGN x 5
  Error: Undefined identifier 'x'

Final state:
```



```
INTEGER a
INTEGER a
HLT
```

```
angelageorge@Angelas-MacBook-Pro-2 project2 % ruby ppl.rb test.ppl

Runtime error at line 2:
  >> INTEGER a
  Error: Identifier 'a' already declared

Final state:
a (int) = 0
```



```
INTEGER a
LIST L
ASSIGN a 5
MERGE L a
HLT
```

```
angelageorge@Angelas-MBP-2 project2 % ruby ppl.rb test.ppl

Runtime error at line 4:
  >> MERGE L a
  Error: MERGE target 'a' is not a LIST

Final state:
a (int) = 5
L (list) = []
```

9. Future Enhancements

- Add labels and symbolic jumps.
- Support variable assignments and basic arithmetic expressions.
- Implement better error handling and debugging mode.
- Optimize list copy performance.
- Extend data types and add built-in list utilities.