Burak Kocausta

1901042605

CSE 341 – HW3

GPP Interpreter Manual

Explanation and test results of lisp and yacc parts, most of the explanations made in list part for types etc.

!! note: Explist uses progn keyword to indicate the expression list, purpose is differing explist from function calls.

1- LISP PART

Shift-reduce parsing algorithm is used when implementing interpreter in the lisp. It builds the evaluation tree with bottom-up parsing. It is not quite advanced, because I did not use parsing table, and most of the checks, evaluations done with manual complex algorithms. Grammar rules are defined, and association is done with grammar checks, I held the rules in a list. My main approach is implementing a left recursive grammar with shift reduce parsing. Since it is not advanced as other shift reduce algorithms, there might be errors for some cases, but I tested as much as I can and shared the results in this report.

I used symbol table to hold variables, and function table to hold function names, argument list, and body. Each function has its own local symbol table when function call occurs. When the function call occurs reference environment has the variable scope of that functions local symbol table, but it uses global function table, since function definitions are global.

Type: The only type in the language is VALUEF.

```
>> 5f2

5f2

>> 3f1

3f1

>> 0f1

0f1

>> 2f2

2f2

>> 5f6

5f6
```

```
>> 5
LEXICAL ERROR: "5" cannot be tokenized
```

f is used as fraction. Integer and other types are not recognized by this language. Both sides of f are unsigned integers. Language does operations with using nominator denominator of fraction.

Assignment Expression: set, and defvar used for assignment expression. Language uses symbol table for association.

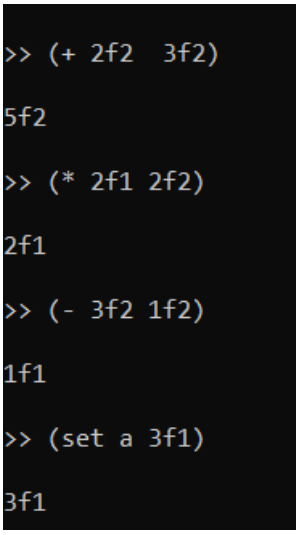
```
>> (set a 2f2)

2f2

>> a

2f2

>> (defvar b 3f5)

3f5

>> b

3f5

>> (set c a)

2f2

>> c

2f2
```

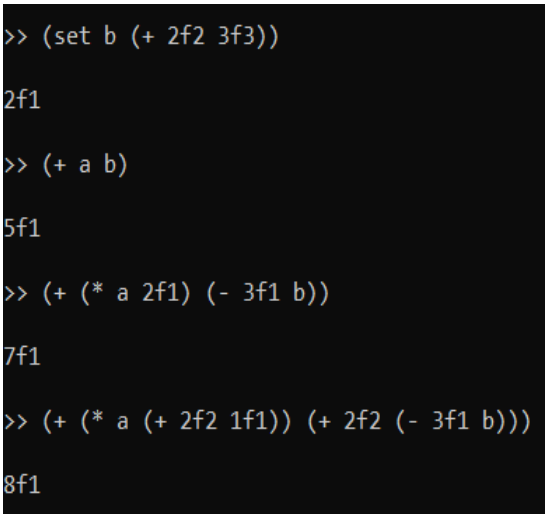
```
>> my_id
Syntax Error: my_id is not defined
```

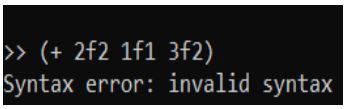
```
>> (set a 3f2)

3f2

>> a

3f2

>> (set a 5f3)

5f3

>> a

5f3
```

If identifier is not defined, gives syntax error. If it is assigned to an already defined identifier, updates the value from symbol table.

Arithmetic Expressions (+ - *): These three operations are supported by the language, it also supports nested operations. Result is simplified after the calculation. Expression returns simplified results. All these 3 operators, can take only 2 operand.

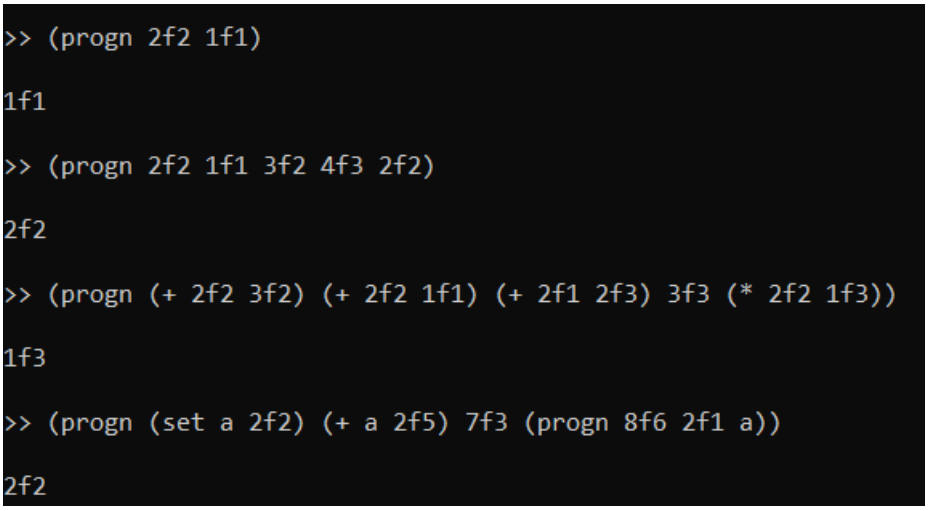






3rd screenshot indicates error when there are more two operands.

Expression List: Expression List requires a **progn** keyword to indicate if it is expression list or not. In this way it can be differentiated from function calls by the interpreter.



```
>> (+ (* (progn 2f2 3f3 (+ 2f2 1f3)) (progn 2f1 7f6 (- 3f1 2f3))) 5f4)

157f36

>> (if true (progn 2f2 (+ 5f3 1f2)) (progn 7f2 1f1 3f3))

13f6

>> (progn (set b 2f3) (+ a b))

5f3
```

If and Boolean Expressions: If uses boolean expression for decision. Boolean expressions, returns invalid syntax when it is directly written to the interpreter without if expressions according to the pdf. Internally it uses true, and false keywords, true and false keywords are also a boolean expression. If expression can take single expression or expression list.

```
>> (if true 3f2 2f2)

3f2

>> (if false 3f2 1f1)

1f1

>> (if (eq 2f2 4f2) 3f3 2f5)

2f5

>> (if (gt 2f1 1f1) (+ 1f2 5f3) (- 3f1 1f1))

13f6
```

```
>> (if (not true) (progn 2f2 1f6 (* 3f2 1f1)) (progn 2f2 5f4 3f2 1f1))

1f1

>> (set my_id (if (eq 3f2 2f2) (+ 5f6 2f1) (- 3f2 1f1)))

1f2

>> my_id

1f2

>> (set my_id2 (if (not (eq my_id 1f2)) 5f3 my_id))

1f2
```

```
>> my_id2

1f2

>> (if (eq my_id my_id2) (progn (set k 3f2) (+ k 2f3)) 0f1)

13f6

>> exit

Terminating Bye.
```

Function Definition and Function Call: Function def requires deffun keyword id, arglist, and explist. After the definition it is associated with the function table. Each function call creates a local stack, variables are different from global symbol table. Each function call uses its local symbol table as referencing enviroment for variables. They can call other functions.

```
>> (deffun sum (x y) (progn (+ x y)))

>> (sum 2f1 3f1)

5f1

>> (set a (sum 2f1 3f1))

5f1

>> (set b (sum 5f2 3f2))

4f1
```

```
>> (set temp (sum a b))

9f1

>> (deffun mult_3 (x y z) (progn (* x (* y z))))

>> (mult_3 a b temp)

180f1
```

```
>> (deffun is_equal (x y) (progn (if (eq x y) 1f1 0f1)))

>> (is_equal 2f2 1f1)

1f1

>> (is_equal 2f1 1f1)

0f1
```

```
>> (deffun find_greatest (x y z) (progn (if (gt x y) (if (gt x z) x z) (if (gt y z) y z))))

>> (find_greatest 3f1 2f1 1f1)

3f1
```

```
>> (if (eq (find_greatest 3f1 2f1 1f1) 3f2) (sum 4f2 3f2) 1f1)
1f1
>> (if (eq (find_greatest 3f1 2f1 1f1) 3f1) (sum 3f2 3f2) 1f1)
3f1
```

```
>> (set my_var 2f7)
2f7
>> (set my_var2 5f3)
5f3
>> (sum my_var my_var2)
41f21
>> exit
Terminating Bye.
```

Functions calling functions

```
>> (deffun fun () (progn (sum 2f2 3f3)))

>> (deffun sum (x y) (progn (+ x y)))

>> (fun)
2f1
```

```
>> (deffun mult_add (x y) (progn (sum (* x y) (+ x y))))

>> (mult_add 2f1 1f1)
5f1
```

2- YACC PART

Type:

```
3f2
Syntax OK.
Result: 3f2
2f1
Syntax OK.
Result: 2f1
5f6
Syntax OK.
Result: 5f6
4f2
Syntax OK.
Result: 4f2
2
LEXICAL ERROR:  2 cannot be tokenized
```

Arithmetic Expressions (+ - *):

```
(+ 2f2 3f1)
Syntax OK.
Result: 4f1
(- 3f2 2f2)
Syntax OK.
Result: 1f2
(* 4f2 1f1)
Syntax OK.
Result: 2f1
(+ (* 3f2 1f2) (- 2f2 1f2))
Syntax OK.
Result: 5f4
(+ (* (+ 2f2 1f1) 2f2) (- 3f2 1f2))
Syntax OK.
Result: 3f1
(+ 2f2 2f2 2f2)
syntax error occured: 2f2
```

Set Operator:

```
(set a 2f2)
Syntax OK.
Result: 2f2
a
Syntax OK.
Result: 2f2
(defvar b 3f5)
Syntax OK.
Result: 3f5
b
Syntax OK.
Result: 3f5
(set c a)
Syntax OK.
Result: 2f2
```

```
c
Syntax OK.
Result: 2f2
(set a 3f2)
Syntax OK.
Result: 3f2
a
Syntax OK.
Result: 3f2
(set a 5f3)
Syntax OK.
Result: 5f3
a
Syntax OK.
Result: 5f3
```

```
(set c (+ a b))
Syntax OK.
Result: 34f15
```

If Expressions and Boolean Expressions:

```
(set a 3f2)
Syntax OK.
Result: 3f2
(set b (if (gt a 2f2) 2f1 5f3))
Syntax OK.
Result: 2f1
(set c (if (eq a b) (progn (* 3f2 1f2) (- 3f2 1f2)) (progn 2f5 3f6 7f2)))
Syntax OK.
Result: 7f2
(if (gt c a) (progn 3f1 2f2) 2f1)
Syntax OK.
Result: 2f2
(if (not (eq a b)) 3f5 (progn (if true 2f2 1f4)))
Syntax OK.
Result: 3f5
(if (gt a b) (progn (if false 2f2 5f3)) (progn 2f6 7f2 8f4))
Syntax OK.
Result: 8f4
```

Expression List:

```
(set a 3f5)
Syntax OK.
Result: 3f5
(progn (+ 3f2 2f2) (set a 2f1) (+ a 3f1))
Syntax OK.
Result: 5f1
a
Syntax OK.
Result: 2f1
(progn 2f2 3f3 1f1 7f2 8f3)
Syntax OK.
Result: 8f3
(progn (+ 2f2 3f4) (set b a) (* a b))
Syntax OK.
Result: 4f1
(if true (progn 2f2 3f4) (progn (+ 3f2 7f6) (+ 5f3 4f1)))
Syntax OK.
Result: 3f4
```

Function Definition:

```
(deffun fun (a b) (progn 2f2 (+ 2f2 1f1) (* 2f2 1f1)))
Syntax OK.
Result: 0f1
```

```
(deffun fun () (progn 4f2 2f2 (+ 3f3 7f6)))
Syntax OK.
Result: 0f1
(deffun fun2 (x y) (progn 4f2 1f1 2f3))
Syntax OK.
Result: 0f1
```

Some of the Errors:

```
(++ 2f2 3f3)
syntax error occured: (null)
```

```
2f2 + 3f3
Syntax OK.
Result: 2f2
syntax error occured: 2f2
```

```
(+ 2f2 3f3 3f3)
syntax error occured: 3f3
```

Compilation & run:

```
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse341/homework/hw3/src/yacc$ make
flex gpp_lexer.l
yacc -d gpp_interpreter.y
cc lex.yy.c y.tab.c -o gpp_interpreter
burak@LAPTOP-7FLC2OAS:/mnt/c/Users/burak kocausta/Desktop/cse341/homework/hw3/src/yacc$ make run
./gpp_interpreter
```