

Automata Formal Languages and Logic
Assignment

UE22CS243A

Syntax Validator of R Programming Language

3rd Semester, Academic Year 2023

Name: Viona Sequeira

SRN: PES2UG22CS665

Name: Vijayalaxmi S.H

SRN: PES2UG22CS663

SECTION: K

Constructs

1) For loop

Lex File

```
AFLL > for > for_lexer.py > ...
1  import ply.lex as lex
2  tokens = ('FOR',
3            'LBRACE',
4            'COLON',
5            'IN',
6            'RBRACE',
7            'LFLOWER',
8            'RFLOWER',
9            'ID',
10           'NUM',
11           'ARROW'
12          )
13
14  def t_FOR(t):
15      r'for'
16      return t
17  def t_IN(t):
18      r'in'
19      return t
20
21  t_LBRACE = r'\('
22  t_RBRACE = r'\)'
23  t_LFLOWER = r'\{'
24  t_RFLOWER = r'\}'
25
26  def t_COLON(t):
27      r'\:'
28      return t
29
30  def t_ARROW(t):
31      r'\<-'
32      return t
```

```
33
34 ✓ def t_ID(t):
35     r'\b([a-zA-Z_][a-zA-Z_0-9]*)\b'
36     return t
37 ✓ def t_NUM(t):
38     r'[0-9][0-9]*'
39     return t
40
41 t_ignore = ' \t'
42
43 ✓ def t_error(t):
44     print(f"Illegal character found {t.value[0]}")
45     t.lexer.skip(1)
46 lexer = lex.lex()
47 data = input()
48 lexer.input(data)
49 ✓ while(1):
50     tok = lexer.token()
51 ✓     if not tok:
52         break
53     print(tok)
```

Yacc file

```
© AFL > for > for_parser.py > p_while
1  import ply.yacc as yacc
2  from for_lexer import tokens
3  flag = 0
4  def p_while(p):
5      ...
6      for_statement : FOR LBRACE ID IN NUM COLON NUM RBRACE LFLOWER statements RFLOWER
7                    | FOR LBRACE ID IN ID RBRACE LFLOWER statements RFLOWER
8                    | FOR LBRACE ID IN NUM COLON NUM RBRACE singleStatement
9                    | FOR LBRACE ID IN ID RBRACE singleStatement
10     ...
11     # this works for
12     #case 1:for(x in 1:10){a<-10}
13     #case 2:for(x in abc){a<-10}
14     #case 3:for(x in 1:10)a<-10
15     #case 4:for(x in abc)a<-10
16
17     # if len(p) == 12:
18     #     p[0] = (p[1], p[3], p[4], p[5], p[6], p[7], p[10])
19     # elif len(p) == 8:
20     #     p[0] = (p[1], p[3], p[4], p[5], p[7])
21     # elif len(p) == 10 and p[6] == ':':
22     #     p[0] = (p[1], p[3], p[4], p[5], p[6], p[7], p[9])
23     # else:
24     #     p[0] = (p[1], p[3], p[4], p[5], p[8])
25  def p_statements(p):
26      ...
27      statements : statements statement
28                | statement
29      ...
30      # if len(p) == 2:
31      #     p[0] = (p[1],)
32      # else:
33      #     p[0] = p[1]+(p[2],)
34  def p_statement(p):
35      ...
36      statement : list
37                | for_statement
38                | empty
39      ...
```

```

40 def p_singleStatement(p):
41     """
42     singleStatement : list
43                     | empty
44                     | for_statement
45     """
46
47 def p_list(p):
48     """
49     list : ID list
50         | ID
51         | ID ARROW NUM
52     """
53
54 def p_empty(p):
55     """
56     empty :
57     """
58     # p[0] = None
59
60 def p_error(p):
61     print("Syntax error",p)
62     global flag
63     flag = 1
64
65 print("Welcome,You are entering for loop declaration")
66 parser = yacc.yacc()
67 while True:
68     flag = 0
69     try:
70         s = input('enter the conditional statement:')
71     except EOFError:
72         break
73     if not s:
74         flag = 0
75         continue
76     result = parser.parse(s)
77     if flag == 0:
78         print("Valid syntax")
79         print("Result:", result)

```

Valid Test Case

```

DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS

PS C:\Users\hp\Desktop\AFL\> cd "c:\Users\hp\Desktop\AFL\for"
PS C:\Users\hp\Desktop\AFL\for> python -u "c:\Users\hp\Desktop\AFL\for.py"
LexToken(ID,'f',1,0)
Welcome,You are entering for loop declaration
enter the conditional statement:for(x in 1:10){a<-10}
Valid syntax
Result: None
enter the conditional statement:

```

Invalid Test Case

```

Result: None
enter the conditional statement:for{x in 1:10}(a<-10)
Syntax error LexToken(LFLOWER,'{',1,3)
enter the conditional statement:

```

2)If else

Lex file

```
AFLL > if else > if_lexer.py > ...
1  import ply.lex as lex
2  tokens=('IF','LEFTBRACKET','RIGHTBRACKET','RIGHTBRACE','LEFTBRACE','ELSE','ID','LESSER',
3         'GREATER',
4         'EQUALS',
5         'NOT',
6         'AND',
7         'OR',
8         'ARROW')
9
10 #defining tokens
11 t_LEFTBRACKET = r'\('
12 t_RIGHTBRACKET=r'\)'
13 t_RIGHTBRACE=r'\}'
14 t_LEFTBRACE=r'\{'
15
16 def t_IF(t):
17     r'if'
18     return t
19
20 def t_ARROW(t):
21     r'<-'
22     return t
23
24 t_ignore = ' \t'
25
26 def t_ELSE(t):
27     r'else'
28     return t
29
30 def t_ID(t):
31     r'\b([a-zA-Z_][a-zA-Z_0-9]*)\b |\b(\d+)\b'
32     return t
```

```
t_LESSER = r'<'
t_GREATER = r'>'
t_EQUALS = r'=(=)?'
t_NOT = r'!'
t_AND = r'&&'
t_OR = r'\|\|'

def t_error(t):
    print(f"Illegal character encountered {t.value[0]}")
    t.lexer.skip(1)

lexer=lex.lex()
data=input()
lexer.input(data)
while(1):
    tok=lexer.token()
    if not tok:
        break
    print(tok)
```

Yacc file

```
AFLL > if else > if_parser.py > ...
1 import ply.yacc as yacc
2 from if_lexer import tokens
3 flag=0
4
5 def p_ifstmt(p):
6     ...
7     ifstmt : IF LEFTBRACKET conditions RIGHTBRACKET LEFTBRACE statements RIGHTBRACE
8             | IF LEFTBRACKET conditions RIGHTBRACKET statementSingle
9             | IF LEFTBRACKET conditions RIGHTBRACKET LEFTBRACE statements RIGHTBRACE ELSE LEFTBRACE statements RIGHTBRACE
10            | IF LEFTBRACKET conditions RIGHTBRACKET statements ELSE statements
11     ...
12     #this works for
13     #case 1: if(a>10){a<-11}
14     #case 2: if(a>10) a<-11
15     #case 3: if(a>10){a<-11}else{a<-10}
16     #case 4: if(a>10)a<-11 else a<-10
17
18     # if len(p) == 6:
19     #     p[0] = (p[1],p[3],p[5])
20     # elif len(p)==11:
21     #     p[0] = (p[1],p[3],p[6],p[8],p[10])
22     # elif len(p)==8 and p[6]=='LEFTBRACE':
23     #     p[0]=(p[1],p[3],p[6])
24     # else:
25     #     p[0]=(p[1],p[3],p[5],p[6],p[7])
26
27 def p_statements(p):
28     ...
29     statements : statements statement
30               | statement
31     ...
32     # if len(p) == 2:
33     #     p[0] = (p[1],)
34     # else:
35     #     p[0] = p[1]+(p[2],)
36
37
38
```

```
AFLL > if else > if_parser.py > ...
39 def p_statement(p):
40     ...
41     statement : list
42               | ifstmt
43               | empty
44     ...
45     # p[0] = (p[1],) if len(p) == 2 else p[1]
46
47 def p_statementSingle(p):
48     ...
49     statementSingle : ifstmt
50                     | list
51                     | empty
52     ...
53     # if len(p) == 3:
54     #     p[0] = (p[1],)
55     # else:
56     #     p[0] = p[1]
57
58 def p_list(p):
59     ...
60     list : ID list
61           | ID
62           | ID ARROW ID
63     ...
64     # if len(p) == 2:
65     #     p[0] = [p[1]]
66     # else:
67     #     p[0] = [p[1]] + p[2]
68
69 def p_empty(p):
70     ...
71     empty :
72     ...
73     # p[0] = None
74
```

```

def p_conditions(p):
    ...
    conditions : ID EQUALS ID
               | ID GREATER ID
               | ID LESSER ID
               | ID GREATER EQUALS ID
               | ID LESSER EQUALS ID
               | ID NOT EQUALS ID
               | conditions AND conditions
               | conditions OR conditions
               | ID
    ...

    # if len(p) == 2:
    #     p[0] = ('condition',p[1])
    # else:
    #     p[0] = ('condition',(p[1],p[2],p[3]))

def p_error(p):
    print("Syntax error")
    global flag
    flag = 1

parser=yacc.yacc()
while True:
    flag=0
    try:
        s=input('enter the declaration:')
    except EOFError:
        break
    if not s:
        flag=0
        continue
    result=parser.parse(s)
    if flag==0:
        print("Result:",result)
        print("VALID SYNTAX")

```

Valid Test Case

```

DEBUG CONSOLE  OUTPUT  TERMINAL  PORTS

KeyboardInterrupt
PS C:\Users\hp\Desktop\AFLL\for> cd "c:\Users\hp\Desktop\AFLL"
PS C:\Users\hp\Desktop\AFLL\if else> python -u "c:\Users\hp\Desktop\AFLL\if else.py"
t
LexToken(ID,'t',1,0)
enter the declaration:if(x>10){a<-10}
Result: None
VALID SYNTAX
enter the declaration:

```

Invalid Test Case

```

enter the declaration:if(){a<-10}
Syntax error
enter the declaration:

```


3) Next

Lex file

```
AFL > next > next_lexer.py > ...
1  import ply.lex as lex
2
3  tokens = ('FOR',
4            'LBRACKET', 'RBRACKET', 'NUM',
5            'LFLOWER', 'RFLOWER', 'NEXT', 'COLON',
6            'ID', 'IN', 'ARROW')
7
8  #defining tokens
9  t_LBRACKET = r'\('
10 t_RBRACKET = r'\)'
11 t_RFLOWER = r'\}'
12 t_LFLOWER = r'\{'
13
14 def t_FOR(t):
15     r'for'
16     return t
17
18 def t_COLON(t):
19     r'\:'
20     return t
21
22 def t_NEXT(t):
23     r'next'
24     return t
25
26 def t_ARROW(t):
27     r'\<-'
28     return t
29
30 def t_NUM(t):
31     r'[0-9][0-9]*'
32
33
34
35
36
37 t_ignore = ' \t'
38
39
40 def t_ID(t):
41     r'\b([a-zA-Z_][a-zA-Z_0-9]*)\b | \b(\d+)\b'
42     return t
43
44 def t_error(t):
45     print(f"illegal character encountered {t.value[0]}")
46     t.lexer.skip(1)
47
48 lexer = lex.lex()
49
50 data = input()
51 lexer.input(data)
52 while(1):
53     tok = lexer.token()
54     if not tok:
55         break
56     print(tok)
57
58
```

Yacc file

```
AFLL > next > next_parser.py > ...
1  import ply.yacc as yacc
2  from next_lexer import tokens
3  from next_lexer import data
4  flag=0
5
6  def p_nextstmt(p):
7      ...
8      nextstmt :  FOR LBRACKET ID IN NUM COLON NUM RBRACKET LFLOWER NEXT statements RFLOWER
9                  |  FOR LBRACKET ID IN ID RBRACKET LFLOWER NEXT statements RFLOWER
10                 |  FOR LBRACKET ID IN NUM COLON NUM RBRACKET NEXT singleStatement
11                 |  FOR LBRACKET ID IN ID RBRACKET NEXT singleStatement
12      ...
13
14      #this works for
15      #case 1: for(x in 1:10){next a<-10}
16      #case 2: for(x in abc){next a<-10}
17      #case 3: for(x in 1:10)next a<-10
18      #case 4: for(x in abc) next a<-10
19
20      # if (len(p) == 8 and p[1]=='IF'):
21      #     p[0] = (p[1],p[3],p[6],p[7])
22      # elif (len(p)==13 and p[0]=='FOR'):
23      #     p[0]= (p[1],p[3],p[4],p[5],p[6],p[7],p[10],p[11])
24      # elif len(p) == 9 and p[0]=='FOR':
25      #     p[0] = (p[1], p[3], p[4], p[5], p[7],p[8])
26      # elif len(p) == 11 and p[6] == ':' and p[0]=='FOR':
27      #     p[0] = (p[1], p[3], p[4], p[5], p[6], p[7], p[9],p[10])
28      # elif (p[1]=='FOR'):
29      #     p[0] = (p[1], p[3], p[4], p[5], p[8],p[9])
30      # elif len(p) == 9 and p[0]=='WHILE':
31      #     p[0] = (p[1],p[3],p[6],p[7])
32      # else:
```

```

AFL > next > next_parser.py > ...
35 def p_statements(p):
36     ...
37     statements : statements statement
38     |          | statement
39     ...
40     # if len(p) == 2:
41     #     p[0] = (p[1],)
42     # else:
43     #     p[0] = p[1]+(p[2],)
44
45 def p_statement(p):
46     ...
47     statement : list
48     |         | nextstmt
49     |         | empty
50     ...
51     # p[0] = (p[1],) if len(p) == 2 else p[1]
52
53 def p_singleStatement(p):
54     ...
55     singleStatement : list
56     |               | empty
57     |               | nextstmt
58     ...
59     # if len(p) == 3:
60     #     p[0] = (p[1],)
61     # else:
62     #     p[0] = p[1]
63
64 def p_list(p):
65     ...
66     list : ID list
67     |    | ID
68     |    | ID ARROW NUM
69     ...
70     # if len(p) == 2:
71     #     p[0] = [p[1]]
72     # else:
73     #     p[0] = [p[1]] + p[2]
74

```

```

74
75 def p_empty(p):
76     ...
77     empty :
78     ...
79     # p[0] = None
80
81
82 def p_error(p):
83     print("Syntax error")
84     global flag
85     flag = 1
86
87 parser=yacc.yacc()
88 while True:
89     flag=0
90     try:
91         s=input('enter the declaration:')
92     except EOFError:
93         break
94     if not s:
95         flag=0
96         continue
97     result=parser.parse(s)
98     if flag==0:
99         print("Result:",result)
100         print("VALID SYNTAX")
101

```

Valid Test Case

```

LexToken(ID,'e',1,0)
enter the declaration:for(x in 1:10){next a<-10}
Result: None
VALID SYNTAX
enter the declaration:

```

Invalid Test Case

```

enter the declaration:for(x in 1:10){next
Syntax error
enter the declaration:

```

4) Repeat

Lex file

```
AFL > repeat > repeat_lex.py > t_ARROW
1  import ply.lex as lex
2  tokens = ('REPEAT',
3           'LBRACKET', 'RBRACKET',
4           'LESSER',
5           'GREATER',
6           'NOT',
7           'AND',
8           'OR',
9           'EQUALS',
10          'LFLOWER',
11          'RFLOWER',
12          'ID', 'BREAK', 'IF', 'ARROW')
13
14  #defining tokens
15  t_LBRACKET = r'\('
16  t_RBRACKET = r'\)'
17  t_RFLOWER = r'\}'
18  t_LFLOWER = r'\{'
19
20  def t_ARROW(t):
21      r'\<-'
22      return t
23
24  def t_REPEAT(t):
25      r'repeat'
26      return t
27
28  def t_IF(t):
29      r'if'
30      return t
31
32  t_ignore = ' \t'
```

```
AFL > repeat > repeat_lex.py > t_ARROW
33
34  def t_BREAK(t):
35      r'break'
36      return t
37
38  def t_ID(t):
39      r'\b([a-zA-Z_][a-zA-Z_0-9]*)\b | \b(\d+)\b'
40      return t
41
42  t_LESSER = r'<'
43  t_GREATER = r'>'
44  t_EQUALS = r'=(=)?'
45  t_NOT = r'!'
46  t_AND = r'&&'
47  t_OR = r'\|\|'
48
49  #defining errors
50  def t_error(t):
51      print(f"illegal character encountered {t.value[0]}")
52      t.lexer.skip(1)
53
54  lexer = lex.lex()
55
56  data = input()
57  lexer.input(data)
58
59  while(1):
60      tok = lexer.token()
61      if not tok:
62          break
63      print(tok)
```

Yacc file

```
AFLL > repeat > repeat_parser.py > p_statements
1  import ply.yacc as yacc
2  from repeat_lexer import tokens
3  from repeat_lexer import data
4  flag=0
5
6  def p_repeatstmt(p):
7      ...
8      repeatstmt : REPEAT LFLOWER statements IF LBRACKET condition RBRACKET LFLOWER BREAK RFLOWER RFLOWER
9      ...
10     #this works for
11     #case 1: repeat{a<-10 if(x>10){break}}
12     # if len(p) == 6:
13     #     p[0] = (p[1],p[3],p[5])
14     # else:
15     #     p[0] = (p[1],p[3],p[6])
16
17  def p_statements(p):
18      ...
19      statements : statements statement
20      | statement
21      ...
22     # if len(p) == 2:
23     #     p[0] = (p[1],)
24     # else:
25     #     p[0] = p[1]+(p[2],)
26
27  def p_statement(p):
28      ...
29      statement : list
30      | repeatstmt
31      | empty
32      ...
33     # p[0] = (p[1],) if len(p) == 2 else p[1]
34
35  def p_list(p):
36      ...
37      list : ID list
38      | ID
39      | ID ARROW ID
40      ...
```

```

AFL > repeat > repeat_parser.py > p_statements
41     # if len(p) == 2:
42     #     p[0] = [p[1]]
43     # else:
44     #     p[0] = [p[1]] + p[2]
45
46 def p_empty(p):
47     ...
48     empty :
49     ...
50     # p[0] = None
51
52 def p_condition(p):
53     ...
54     condition : ID EQUALS ID
55                | ID GREATER ID
56                | ID LESSER ID
57                | ID GREATER EQUALS ID
58                | ID LESSER EQUALS ID
59                | ID NOT EQUALS ID
60                | condition AND condition
61                | condition OR condition
62                | ID
63     ...
64     # if len(p) == 2:
65     #     p[0] = ('condition',p[1])
66     # else:
67     #     p[0] = ('condition',(p[1],p[2],p[3]))
68
69 def p_error(p):
70     print("Syntax error")
71     global flag
72     flag = 1
73
74 parser=yacc.yacc()
75 while True:
76     flag=0
77     try:
78         s=input('enter the declaration:')
79     except EOFError:

```

```

57 | ID GREATER EQUALS ID
58 | ID LESSER EQUALS ID
59 | ID NOT EQUALS ID
60 | condition AND condition
61 | condition OR condition
62 | ID
63 | ...
64 | # if len(p) == 2:
65 | #     p[0] = ('condition',p[1])
66 | # else:
67 | #     p[0] = ('condition',(p[1],p[2],p[3]))
68 |
69 | def p_error(p):
70 |     print("Syntax error")
71 |     global flag
72 |     flag = 1
73 |
74 | parser=yacc.yacc()
75 | while True:
76 |     flag=0
77 |     try:
78 |         s=input('enter the declaration:')
79 |     except EOFError:
80 |         break
81 |     if not s:
82 |         flag=0
83 |         continue
84 |     result=parser.parse(s)
85 |     if flag==0:
86 |         print("Result:",result)
87 |         print("VALID SYNTAX")
88 |

```

Valid Test Case

```

LexToken(RFLOWER, '}', 1, 28)
enter the declaration:repeat{a<-10 if(x>10){break}}
Result: None
VALID SYNTAX
enter the declaration:

```

Invalid Test Case

```

enter the declaration:repet{a<-10}
Syntax error
enter the declaration:

```


5) While loop

Lex file

```
AFLL > while > while_lexer.py > t_ARROW
1  import ply.lex as lex
2  tokens = ('WHILE',
3            'LBRACKET', 'RBRACKET',
4            'LESSER',
5            'GREATER',
6            'NOT',
7            'AND',
8            'OR',
9            'EQUALS',
10           'LFLOWER',
11           'RFLOWER',
12           'ID', 'ARROW')
13
14  #Defining token rules
15  t_LBRACKET = r'\('
16  t_RBRACKET = r'\)'
17  t_LFLOWER = r'\{'
18  t_RFLOWER = r'\}'
19  def t_WHILE(t):
20      r'while'
21      return t
22  def t_ID(t):
23      r'\b([a-zA-Z_][a-zA-Z_0-9]*)\b|\b(\d+)\b'
24      return t
25  def t_ARROW(t):
26      r'\<-'
27      return t
28  t_LESSER = r'<'
29  t_GREATER = r'>'
30  t_EQUALS = r'=(=)?'
31  t_NOT = r'!'
32  t_AND = r'&&'
33
34  return t
35
36  t_LESSER = r'<'
37  t_GREATER = r'>'
38  t_EQUALS = r'=(=)?'
39  t_NOT = r'!'
40  t_AND = r'&&'
41  t_OR = r'\|\|'
42  t_ignore = ' \t'
43
44  #Incase of error
45  def t_error(t):
46      print(f"Illegal character found {t.value[0]}")
47      t.lexer.skip(1)
48
49  lexer = lex.lex()
50  data = input()
51  lexer.input(data)
52  while(1):
53      tok = lexer.token()
54      if not tok:
55          break
56      print(tok)
```

Yacc file

```
© AFL > while > while_parser.py > ...
1  import ply.yacc as yacc
2  from while_lexer import tokens
3  from while_lexer import data
4  flag = 0
5  def p_while(p):
6      ...
7      while_statement : WHILE LBRACKET conditions RBRACKET LFLOWER statements RFLOWER
8                      | WHILE LBRACKET conditions RBRACKET singleStatement
9      ...
10     #this works for
11     #case 1: while(x>10){a<-10}
12     #case 2: while(x>10)a<-10
13
14     # if len(p) == 6:
15     #     p[0] = (p[1],p[3],p[5])
16     # else:
17     #     p[0] = (p[1],p[3],p[6])
18 def p_statements(p):
19     ...
20     statements : statements statement
21               | statement
22     ...
23     # if len(p) == 2:
24     #     p[0] = (p[1],)
25     # else:
26     #     p[0] = p[1]+(p[2],)
27 def p_statement(p):
28     ...
29     statement : list
30              | while_statement
31              | empty
32     ...
33     # if len(p) == 3:
34     #     p[0] = (p[1],)
35     # else:
36     #     p[0] = p[1]
37 def p_singleStatement(p):
38     ...
39     singleStatement : list
```

● AFL > while > while_parser.py > ...

```
37 def p_singleStatement(p):
38     ...
39     singleStatement : list
40     | empty
41     | while_statement
42     ...
43     # if len(p) == 3:
44     #     p[0] = (p[1],)
45     # else:
46     #     p[0] = p[1]
47 def p_list(p):
48     ...
49     list : ID list
50     | ID
51     | ID ARROW ID
52     ...
53     # if len(p) == 2:
54     #     p[0] = [p[1]]
55     # else:
56     #     p[0] = [p[1]]+p[2]
57
58 def p_empty(p):
59     ...
60     empty :
61     ...
62     # p[0] = None
63 def p_conditions(p):
64     ...
65     conditions : ID EQUALS ID
66     | ID GREATER ID
67     | ID LESSER ID
68     | ID GREATER EQUALS ID
69     | ID LESSER EQUALS ID
70     | ID NOT EQUALS ID
71     | conditions AND conditions
72     | conditions OR conditions
73     | ID
74     ...
```

```

| ID LESSER_EQUALS ID
| ID NOT_EQUALS ID
| conditions AND conditions
| conditions OR conditions
| ID
...
# if len(p) == 2:
#     p[0] = ('condition',p[1])
# else :
#     p[0] = ('condition',p[1],p[2],p[3])
def p_error(p):
    print("Syntax error")
    global flag
    flag = 1
#From here, just copy paste and change the input statement for every other construct
#Don't forget to globally declare flag and also make flag 1 at error
parser = yacc.yacc()
while True:
    flag = 0
    try:
        s = input('enter while statement:')
    except EOFError:
        break
    if not s:
        flag = 0
        continue
    result = parser.parse(s)
    if flag == 0:
        print("Valid syntax")
        print("Result:", result)

```

Valid Test Case

```

LexToken(ID,'r',1,0)
enter while statement:while(x>10){a<-10}
Valid syntax
Result: None
enter while statement:

```

Invalid Test Case

```

enter while statement:while(x>10){a<-10
Syntax error
enter while statement:

```