Parse and CST Testing Results

Emily Doran

Emily.Doran1@Marist.edu

April 5, 2021

Basic Test Cases (Verbose Mode)

Input text:

```
{}$
{{{{{{}}}}}}$
{{{{{}}} /* comments are ignored */ }}}}$
{/* comments are still ignored */  int @}$
{
  int a
  a = a
  string b
  a = b
}$
```

Output:

```
INFO  Lexer - Lexing program 1...
DEBUG Lexer - T_L_BRACE [ { ] found at (1:1)
DEBUG Lexer - T_R_BRACE [ } ] found at (1:2)
DEBUG Lexer - T_EOP [ $ ] found at (1:3)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 1 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: Parse completed successfully

CST for program 1 ...
<Program>
-<Block>
```

```
--[{]
--[StatementList]
--[}]
-[$]



INFO  Lexer - Lexing program 2...
DEBUG Lexer - T_L_BRACE [ { ] found at (2:1)
DEBUG Lexer - T_L_BRACE [ { ] found at (2:2)
DEBUG Lexer - T_L_BRACE [ { ] found at (2:3)
DEBUG Lexer - T_L_BRACE [ { ] found at (2:4)
DEBUG Lexer - T_L_BRACE [ { ] found at (2:5)
DEBUG Lexer - T_L_BRACE [ { ] found at (2:6)
DEBUG Lexer - T_R_BRACE [ } ] found at (2:7)
DEBUG Lexer - T_R_BRACE [ } ] found at (2:8)
DEBUG Lexer - T_R_BRACE [ } ] found at (2:9)
DEBUG Lexer - T_R_BRACE [ } ] found at (2:10)
DEBUG Lexer - T_R_BRACE [ } ] found at (2:11)
DEBUG Lexer - T_R_BRACE [ } ] found at (2:12)
DEBUG Lexer - T_EOP [ $ ] found at (2:13)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 2 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: Parse completed successfully

CST for program 2 ...
<Program>
```

```
-<Block>
--[{]
--<StatementList>
---<Statement>
----<Block>
-----[{]
-----<StatementList>
------<Statement>
-------<Block>
--------[{]
--------<StatementList>
---------<Statement>
----------<Block>
----------[{]
----------<StatementList>
----------<Statement>
----------<Block>
----------[{]
----------<StatementList>
----------<Statement>
----------<Block>
----------[{]
----------[StatementList]
----------[}]
----------[}]
----------[}]
--------[}]
-----[}]
--[}]
-[$]



INFO  Lexer - Lexing program 3...
DEBUG Lexer - T_L_BRACE [ { ] found at (3:1)
DEBUG Lexer - T_L_BRACE [ { ] found at (3:2)
DEBUG Lexer - T_L_BRACE [ { ] found at (3:3)
DEBUG Lexer - T_L_BRACE [ { ] found at (3:4)
DEBUG Lexer - T_L_BRACE [ { ] found at (3:5)
DEBUG Lexer - T_L_BRACE [ { ] found at (3:6)
DEBUG Lexer - T_R_BRACE [ } ] found at (3:7)
DEBUG Lexer - T_R_BRACE [ } ] found at (3:8)
DEBUG Lexer - T_R_BRACE [ } ] found at (3:9)
DEBUG Lexer - T_R_BRACE [ } ] found at (3:38)
DEBUG Lexer - T_R_BRACE [ } ] found at (3:39)
DEBUG Lexer - T_R_BRACE [ } ] found at (3:40)
DEBUG Lexer - T_R_BRACE [ } ] found at (3:41)
DEBUG Lexer - T_EOP [ $ ] found at (3:42)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 3 ...
PARSER: parse()
```

```
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: ERROR: Expected [EOP] got '}' on line 3
PARSER: Parse failed with 1 error(s)

CST for program 3: Skipped due to PARSER error(s)


INFO  Lexer - Lexing program 4...
DEBUG Lexer - T_L_BRACE [ { ] found at (4:1)
DEBUG Lexer - T_VARIABLE_TYPE [ int ] found at (4:36)
ERROR Lexer - Error: 4:40 Unrecognized Token: @
DEBUG Lexer - T_R_BRACE [ } ] found at (4:41)
DEBUG Lexer - T_EOP [ $ ] found at (4:42)
ERROR Lexer - Lex failed with 1 error(s)

PARSER: Skipped due to LEXER error(s)

CST for program 4: Skipped due to LEXER error(s)


INFO  Lexer - Lexing program 5...
DEBUG Lexer - T_L_BRACE [ { ] found at (5:1)
DEBUG Lexer - T_VARIABLE_TYPE [ int ] found at (6:3)
DEBUG Lexer - T_ID [ a ] found at (6:7)
DEBUG Lexer - T_ID [ a ] found at (7:3)
DEBUG Lexer - T_ASSIGN_OP [ = ] found at (7:5)
DEBUG Lexer - T_ID [ a ] found at (7:7)
DEBUG Lexer - T_VARIABLE_TYPE [ string ] found at (8:3)
DEBUG Lexer - T_ID [ b ] found at (8:10)
DEBUG Lexer - T_ID [ a ] found at (9:3)
```

```
DEBUG Lexer - T_ASSIGN_OP [ = ] found at (9:5)
DEBUG Lexer - T_ID [ b ] found at (9:7)
DEBUG Lexer - T_R_BRACE [ } ] found at (10:1)
DEBUG Lexer - T_EOP [ $ ] found at (10:2)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 5 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignStatement()
PARSER: parseExpr()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignStatement()
PARSER: parseExpr()
PARSER: parseStatementList()
PARSER: Parse completed successfully

CST for program 5 ...
<Program>
-<Block>
--[{]
--<StatementList>
---<Statement>
----<VarDecl>
-----<Type>
------[int]
-----<Id>
------[a]
---<StatementList>
----<Statement>
-----<AssignStatement>
------<Id>
-------[a]
------[=]
------<Expression>
-------<Id>
--------[a]
----<StatementList>
-----<Statement>
------<VarDecl>
```

```
-------<Type>
--------[string]
-------<Id>
--------[b]
-----<StatementList>
------<Statement>
-------<AssignStatement>
--------<Id>
---------[a]
--------[=]
--------<Expression>
---------<Id>
----------[b]
--[}]
-[$]
```

Programs 1, 2, and 5 passed both Lex and Parse without throwing any errors, so they output the CSTs for their programs. Program 3 passed Lex, but failed Parse because there is an extra right brace, so when the program expects to find EOP but instead finds the brace, it throws an error and does not output the CST. Program 4 fails Lex because it has an unrecognized token '@' in it, so it does not attempt parsing and thus does not output a CST.

While Loop Test Cases (Verbose Mode)

Input text:

```
{while false{}}$
{
  while (a!=2){
    a = 3+1
  }
}$
{while (c==true){print(a)}}$
{while (b){}}$
{while ("hi" == 2)}}$
```

Output:

```
INFO  Lexer - Lexing program 1...
DEBUG Lexer - T_L_BRACE [ { ] found at (1:1)
DEBUG Lexer - T_WHILE [ while ] found at (1:2)
DEBUG Lexer - T_BOOL_FALSE [ false ] found at (1:8)
DEBUG Lexer - T_L_BRACE [ { ] found at (1:13)
DEBUG Lexer - T_R_BRACE [ } ] found at (1:14)
DEBUG Lexer - T_R_BRACE [ } ] found at (1:15)
DEBUG Lexer - T_EOP [ $ ] found at (1:16)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 1 ...
PARSER: parse()
PARSER: parseProgram()
```

```
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseWhileStatement()
PARSER: parseBooleanExpr()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: Parse completed successfully

CST for program 1 ...
<Program>
-<Block>
--[{]
--<StatementList>
---<Statement>
----<WhileStatement>
-----[while]
-----<BooleanExpression>
------<BoolVal>
-------[false]
-----<Block>
------[{]
------[StatementList]
------[}]
--[}]
-[$]



INFO  Lexer - Lexing program 2...
DEBUG Lexer - T_L_BRACE [ { ] found at (2:1)
DEBUG Lexer - T_WHILE [ while ] found at (3:3)
DEBUG Lexer - T_L_PAREN [ ( ] found at (3:9)
DEBUG Lexer - T_ID [ a ] found at (3:10)
DEBUG Lexer - T_INEQUALITY_OP [ != ] found at (3:11)
DEBUG Lexer - T_DIGIT [ 2 ] found at (3:13)
DEBUG Lexer - T_R_PAREN [ ) ] found at (3:14)
DEBUG Lexer - T_L_BRACE [ { ] found at (3:15)
DEBUG Lexer - T_ID [ a ] found at (4:5)
DEBUG Lexer - T_ASSIGN_OP [ = ] found at (4:7)
DEBUG Lexer - T_DIGIT [ 3 ] found at (4:9)
DEBUG Lexer - T_ADDITION_OP [ + ] found at (4:10)
DEBUG Lexer - T_DIGIT [ 1 ] found at (4:11)
DEBUG Lexer - T_R_BRACE [ } ] found at (5:3)
DEBUG Lexer - T_R_BRACE [ } ] found at (6:1)
DEBUG Lexer - T_EOP [ $ ] found at (6:2)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 2 ...
PARSER: parse()
PARSER: parseProgram()
```

```
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseWhileStatement()
PARSER: parseBooleanExpr()
PARSER: parseExpr()
PARSER: parseBoolOp()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignStatement()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: Parse completed successfully

CST for program 2 ...
<Program>
-<Block>
--[{]
--<StatementList>
---<Statement>
----<WhileStatement>
-----[while]
-----<BooleanExpression>
------[(]
------<Expression>
-------<Id>
--------[a]
------<BoolOp>
-------[!=]
------<Expression>
-------<IntegerExpression>
--------<Digit>
---------[2]
------[)]
-----<Block>
------[{]
------<StatementList>
-------<Statement>
--------<AssignStatement>
---------<Id>
----------[a]
---------[=]
---------<Expression>
----------<IntegerExpression>
-----------<Digit>
```

```
------------[3]
-----------<IntOp>
------------[+]
-----------<Expression>
------------<IntegerExpression>
-------------<Digit>
--------------[1]
------[}]
--[}]
-[$]



INFO  Lexer - Lexing program 3...
DEBUG Lexer - T_L_BRACE [ { ] found at (7:1)
DEBUG Lexer - T_WHILE [ while ] found at (7:2)
DEBUG Lexer - T_L_PAREN [ ( ] found at (7:8)
DEBUG Lexer - T_ID [ c ] found at (7:9)
DEBUG Lexer - T_EQUALITY_OP [ == ] found at (7:10)
DEBUG Lexer - T_BOOL_TRUE [ true ] found at (7:12)
DEBUG Lexer - T_R_PAREN [ ) ] found at (7:16)
DEBUG Lexer - T_L_BRACE [ { ] found at (7:17)
DEBUG Lexer - T_PRINT [ print ] found at (7:18)
DEBUG Lexer - T_L_PAREN [ ( ] found at (7:23)
DEBUG Lexer - T_ID [ a ] found at (7:24)
DEBUG Lexer - T_R_PAREN [ ) ] found at (7:25)
DEBUG Lexer - T_R_BRACE [ } ] found at (7:26)
DEBUG Lexer - T_R_BRACE [ } ] found at (7:27)
DEBUG Lexer - T_EOP [ $ ] found at (7:28)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 3 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseWhileStatement()
PARSER: parseBooleanExpr()
PARSER: parseExpr()
PARSER: parseBoolOp()
PARSER: parseExpr()
PARSER: parseBooleanExpr()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parsePrintStatement()
PARSER: parseExpr()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: Parse completed successfully
```

```
CST for program 3 ...
<Program>
-<Block>
--[{]
--<StatementList>
---<Statement>
----<WhileStatement>
-----[while]
-----<BooleanExpression>
------[(]
------<Expression>
-------<Id>
--------[c]
------<BoolOp>
-------[==]
------<Expression>
-------<BooleanExpression>
--------<BoolVal>
---------[true]
------[)]
-----<Block>
------[{]
------<StatementList>
-------<Statement>
--------<PrintStatement>
---------[print]
---------[(]
---------<Expression>
----------<Id>
-----------[a]
---------[)]
------[}]
--[}]
-[$]



INFO  Lexer - Lexing program 4...
DEBUG Lexer - T_L_BRACE [ { ] found at (8:1)
DEBUG Lexer - T_WHILE [ while ] found at (8:2)
DEBUG Lexer - T_L_PAREN [ ( ] found at (8:8)
DEBUG Lexer - T_ID [ b ] found at (8:9)
DEBUG Lexer - T_R_PAREN [ ) ] found at (8:10)
DEBUG Lexer - T_L_BRACE [ { ] found at (8:11)
DEBUG Lexer - T_R_BRACE [ } ] found at (8:12)
DEBUG Lexer - T_R_BRACE [ } ] found at (8:13)
DEBUG Lexer - T_EOP [ $ ] found at (8:14)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 4 ...
PARSER: parse()
PARSER: parseProgram()
```

```
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseWhileStatement()
PARSER: parseBooleanExpr()
PARSER: parseExpr()
PARSER: parseBoolOp()
PARSER: ERROR: Expected BoolOp got ')' on line 8
PARSER: Parse failed with 1 error(s)

CST for program 4: Skipped due to PARSER error(s)


INFO  Lexer - Lexing program 5...
DEBUG Lexer - T_L_BRACE [ { ] found at (9:1)
DEBUG Lexer - T_WHILE [ while ] found at (9:2)
DEBUG Lexer - T_L_PAREN [ ( ] found at (9:8)
DEBUG Lexer - T_QUOTE [ " ] found at (9:9)
DEBUG Lexer - T_CHAR [ h ] found at (9:10)
DEBUG Lexer - T_CHAR [ i ] found at (9:11)
DEBUG Lexer - T_QUOTE [ " ] found at (9:12)
DEBUG Lexer - T_EQUALITY_OP [ == ] found at (9:14)
DEBUG Lexer - T_DIGIT [ 2 ] found at (9:17)
DEBUG Lexer - T_R_PAREN [ ) ] found at (9:18)
DEBUG Lexer - T_R_BRACE [ } ] found at (9:19)
DEBUG Lexer - T_R_BRACE [ } ] found at (9:20)
DEBUG Lexer - T_EOP [ $ ] found at (9:21)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 5 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseWhileStatement()
PARSER: parseBooleanExpr()
PARSER: parseExpr()
PARSER: parseStringExpr()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseBoolOp()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseBlock()
PARSER: ERROR: Expected [{] got '}' on line 9
PARSER: parseStatementList()
PARSER: Parse failed with 1 error(s)

CST for program 5: Skipped due to PARSER error(s)
```

Programs 1, 2, and 3 all pass both Lex and Parse, so they generate and print out the CSTs for the programs.

Program 4 throws a parse error because inside the parenthesis of a while loop, a boolean expression is expected, but instead the program finds the id and looks for the boolOp but doesn't find it. So the program throws the error and discards the CST. Program 5 also throws a parse error (not for the weird "hi" == 2 inside the boolean expression). Although this looks very strange, it is a valid boolean expression because it consists of an Expr boolop Expr. Instead, it throws the parse error because it it missing the left brace in the while loop.

If Statement Test Cases (Verbose Mode)

Input text:

```
{if true {print ("string")}}$

{
  if (a!=b) {
    int x
    boolean y
    x = 3
    y = false
  }
}$

{
  if false {
    int x
    boolean y
    x = 3
    y =
  }
}$

{
  if (x = 3) {
    int x
  }
}$
```

Output:

```
INFO  Lexer - Lexing program 1...
DEBUG Lexer - T_L_BRACE [ { ] found at (1:1)
DEBUG Lexer - T_IF [ if ] found at (1:2)
DEBUG Lexer - T_BOOL_TRUE [ true ] found at (1:5)
DEBUG Lexer - T_L_BRACE [ { ] found at (1:10)
DEBUG Lexer - T_PRINT [ print ] found at (1:11)
DEBUG Lexer - T_L_PAREN [ ( ] found at (1:17)
DEBUG Lexer - T_QUOTE [ " ] found at (1:18)
DEBUG Lexer - T_CHAR [ s ] found at (1:19)
DEBUG Lexer - T_CHAR [ t ] found at (1:20)
DEBUG Lexer - T_CHAR [ r ] found at (1:21)
DEBUG Lexer - T_CHAR [ i ] found at (1:22)
```

```
DEBUG Lexer - T_CHAR [ n ] found at (1:23)
DEBUG Lexer - T_CHAR [ g ] found at (1:24)
DEBUG Lexer - T_QUOTE [ " ] found at (1:25)
DEBUG Lexer - T_R_PAREN [ ) ] found at (1:26)
DEBUG Lexer - T_R_BRACE [ } ] found at (1:27)
DEBUG Lexer - T_R_BRACE [ } ] found at (1:28)
DEBUG Lexer - T_EOP [ $ ] found at (1:29)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 1 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseIfStatement()
PARSER: parseBooleanExpr()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parsePrintStatement()
PARSER: parseExpr()
PARSER: parseStringExpr()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: Parse completed successfully

CST for program 1 ...
<Program>
-<Block>
--[{]
--<StatementList>
---<Statement>
----<IfStatement>
-----[if]
-----<BooleanExpression>
------<BoolVal>
-------[true]
-----<Block>
------[{]
------<StatementList>
-------<Statement>
--------<PrintStatement>
---------[print]
---------[(]
---------<Expression>
```

```
----------<StringExpression>
----------["]
----------<CharList>
----------<Char>
----------[s]
----------<CharList>
----------<Char>
----------[t]
----------<CharList>
----------<Char>
----------[r]
----------<CharList>
----------<Char>
----------[i]
----------<CharList>
----------<Char>
----------[n]
----------<CharList>
----------<Char>
----------[g]
----------["]
---------[)]
------[}]
--[}]
-[$]
```

```
INFO  Lexer - Lexing program 2...
DEBUG Lexer - T_L_BRACE [ { ] found at (3:1)
DEBUG Lexer - T_IF [ if ] found at (4:3)
DEBUG Lexer - T_L_PAREN [ ( ] found at (4:6)
DEBUG Lexer - T_ID [ a ] found at (4:7)
DEBUG Lexer - T_INEQUALITY_OP [ != ] found at (4:8)
DEBUG Lexer - T_ID [ b ] found at (4:10)
DEBUG Lexer - T_R_PAREN [ ) ] found at (4:11)
DEBUG Lexer - T_L_BRACE [ { ] found at (4:13)
DEBUG Lexer - T_VARIABLE_TYPE [ int ] found at (5:5)
DEBUG Lexer - T_ID [ x ] found at (5:9)
DEBUG Lexer - T_VARIABLE_TYPE [ boolean ] found at (6:5)
DEBUG Lexer - T_ID [ y ] found at (6:13)
DEBUG Lexer - T_ID [ x ] found at (7:5)
DEBUG Lexer - T_ASSIGN_OP [ = ] found at (7:7)
DEBUG Lexer - T_DIGIT [ 3 ] found at (7:9)
DEBUG Lexer - T_ID [ y ] found at (8:5)
DEBUG Lexer - T_ASSIGN_OP [ = ] found at (8:7)
DEBUG Lexer - T_BOOL_FALSE [ false ] found at (8:9)
DEBUG Lexer - T_R_BRACE [ } ] found at (9:3)
DEBUG Lexer - T_R_BRACE [ } ] found at (10:1)
DEBUG Lexer - T_EOP [ $ ] found at (10:2)
INFO  Lexer - Lex completed with 0 errors
```

```
PARSER: Parsing program 2 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseIfStatement()
PARSER: parseBooleanExpr()
PARSER: parseExpr()
PARSER: parseBoolOp()
PARSER: parseExpr()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignStatement()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignStatement()
PARSER: parseExpr()
PARSER: parseBooleanExpr()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: Parse completed successfully

CST for program 2 ...
<Program>
-<Block>
--[{]
--<StatementList>
---<Statement>
----<IfStatement>
-----[if]
-----<BooleanExpression>
------[(]
------<Expression>
-------<Id>
--------[a]
------<BoolOp>
-------[!=]
------<Expression>
-------<Id>
--------[b]
```

```
------[)]
-----<Block>
------[{]
------<StatementList>
-------<Statement>
--------<VarDecl>
---------<Type>
----------[int]
---------<Id>
----------[x]
-------<StatementList>
--------<Statement>
---------<VarDecl>
----------<Type>
-----------[boolean]
----------<Id>
-----------[y]
--------<StatementList>
---------<Statement>
----------<AssignStatement>
-----------<Id>
------------[x]
-----------[=]
-----------<Expression>
------------<IntegerExpression>
-------------<Digit>
--------------[3]
---------<StatementList>
----------<Statement>
-----------<AssignStatement>
------------<Id>
-------------[y]
------------[=]
------------<Expression>
-------------<BooleanExpression>
--------------<BoolVal>
---------------[false]
------[}]
--[}]
-[$]



INFO  Lexer - Lexing program 3...
DEBUG Lexer - T_L_BRACE [ { ] found at (12:1)
DEBUG Lexer - T_IF [ if ] found at (13:3)
DEBUG Lexer - T_BOOL_FALSE [ false ] found at (13:6)
DEBUG Lexer - T_L_BRACE [ { ] found at (13:12)
DEBUG Lexer - T_VARIABLE_TYPE [ int ] found at (14:5)
DEBUG Lexer - T_ID [ x ] found at (14:9)
DEBUG Lexer - T_VARIABLE_TYPE [ boolean ] found at (15:5)
DEBUG Lexer - T_ID [ y ] found at (15:13)
```

```
DEBUG Lexer - T_ID [ x ] found at (16:5)
DEBUG Lexer - T_ASSIGN_OP [ = ] found at (16:7)
DEBUG Lexer - T_DIGIT [ 3 ] found at (16:9)
DEBUG Lexer - T_ID [ y ] found at (17:5)
DEBUG Lexer - T_ASSIGN_OP [ = ] found at (17:7)
DEBUG Lexer - T_R_BRACE [ } ] found at (18:3)
DEBUG Lexer - T_R_BRACE [ } ] found at (19:1)
DEBUG Lexer - T_EOP [ $ ] found at (19:2)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 3 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseIfStatement()
PARSER: parseBooleanExpr()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignStatement()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignStatement()
PARSER: parseExpr()
PARSER: ERROR: Expected [IntExpr, StringExpr, BooleanExpr, Id] got '}' on line 18
PARSER: parseStatementList()
PARSER: Parse failed with 1 error(s)

CST for program 3: Skipped due to PARSER error(s)


INFO  Lexer - Lexing program 4...
DEBUG Lexer - T_L_BRACE [ { ] found at (21:1)
DEBUG Lexer - T_IF [ if ] found at (22:3)
DEBUG Lexer - T_L_PAREN [ ( ] found at (22:6)
DEBUG Lexer - T_ID [ x ] found at (22:7)
DEBUG Lexer - T_ASSIGN_OP [ = ] found at (22:9)
DEBUG Lexer - T_DIGIT [ 3 ] found at (22:11)
DEBUG Lexer - T_R_PAREN [ ) ] found at (22:12)
DEBUG Lexer - T_L_BRACE [ { ] found at (22:14)
```

```
DEBUG Lexer - T_VARIABLE_TYPE [ int ] found at (23:5)
DEBUG Lexer - T_ID [ x ] found at (23:9)
DEBUG Lexer - T_R_BRACE [ } ] found at (24:3)
DEBUG Lexer - T_R_BRACE [ } ] found at (25:1)
DEBUG Lexer - T_EOP [ $ ] found at (25:2)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 4 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseIfStatement()
PARSER: parseBooleanExpr()
PARSER: parseExpr()
PARSER: parseBoolOp()
PARSER: ERROR: Expected BoolOp got '=' on line 22
PARSER: Parse failed with 1 error(s)

CST for program 4: Skipped due to PARSER error(s)
```

The first two programs complete without throwing any errors. They have all of the necessary components for if statements and all of the expressions within. The third program fails because we have "y = " and then close the if statement. The parser throws an error because it is expecting an Expression, but finds the closing brace. The fourth program fails because we have "if(x = 3)". This is not a valid boolean expression because the parser finds the id and expects a boolop to come next, but instead finds the assign op, so it fails.

Parse Error Test Cases (Verbose Mode)

Input text:

```
{intaa=3+}$
{int a a=2 print("abc")$
{x = }$
```

Output:

```
INFO  Lexer - Lexing program 1...
DEBUG Lexer - T_L_BRACE [ { ] found at (1:1)
DEBUG Lexer - T_VARIABLE_TYPE [ int ] found at (1:2)
DEBUG Lexer - T_ID [ a ] found at (1:5)
DEBUG Lexer - T_ID [ a ] found at (1:6)
DEBUG Lexer - T_ASSIGN_OP [ = ] found at (1:7)
DEBUG Lexer - T_DIGIT [ 3 ] found at (1:8)
DEBUG Lexer - T_ADDITION_OP [ + ] found at (1:9)
DEBUG Lexer - T_R_BRACE [ } ] found at (1:10)
DEBUG Lexer - T_EOP [ $ ] found at (1:11)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 1 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
```

```
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignStatement()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseExpr()
PARSER: ERROR: Expected [IntExpr, StringExpr, BooleanExpr, Id] got '}' on line 1
PARSER: Parse failed with 1 error(s)

CST for program 1: Skipped due to PARSER error(s)


INFO  Lexer - Lexing program 2...
DEBUG Lexer - T_L_BRACE [ { ] found at (2:1)
DEBUG Lexer - T_VARIABLE_TYPE [ int ] found at (2:2)
DEBUG Lexer - T_ID [ a ] found at (2:6)
DEBUG Lexer - T_ID [ a ] found at (2:8)
DEBUG Lexer - T_ASSIGN_OP [ = ] found at (2:9)
DEBUG Lexer - T_DIGIT [ 2 ] found at (2:10)
DEBUG Lexer - T_PRINT [ print ] found at (2:12)
DEBUG Lexer - T_L_PAREN [ ( ] found at (2:17)
DEBUG Lexer - T_QUOTE [ " ] found at (2:18)
DEBUG Lexer - T_CHAR [ a ] found at (2:19)
DEBUG Lexer - T_CHAR [ b ] found at (2:20)
DEBUG Lexer - T_CHAR [ c ] found at (2:21)
DEBUG Lexer - T_QUOTE [ " ] found at (2:22)
DEBUG Lexer - T_R_PAREN [ ) ] found at (2:23)
DEBUG Lexer - T_EOP [ $ ] found at (2:24)
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 2 ...
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignStatement()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parsePrintStatement()
PARSER: parseExpr()
PARSER: parseStringExpr()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseStatementList()
```

```
 PARSER: parseStatement()
 PARSER: ERROR: Expected [PrintStatement, AssignStatement, VarDecl, WhileStatement,
IfStatement, Block] got '$' on line 2
 PARSER: Parse failed with 1 error(s)

 CST for program 2: Skipped due to PARSER error(s)


 INFO  Lexer - Lexing program 3...
 DEBUG Lexer - T_L_BRACE [ { ] found at (3:1)
 DEBUG Lexer - T_ID [ x ] found at (3:2)
 DEBUG Lexer - T_ASSIGN_OP [ = ] found at (3:4)
 DEBUG Lexer - T_R_BRACE [ } ] found at (3:6)
 DEBUG Lexer - T_EOP [ $ ] found at (3:7)
 INFO  Lexer - Lex completed with 0 errors

 PARSER: Parsing program 3 ...
 PARSER: parse()
 PARSER: parseProgram()
 PARSER: parseBlock()
 PARSER: parseStatementList()
 PARSER: parseStatement()
 PARSER: parseAssignStatement()
 PARSER: parseExpr()
 PARSER: ERROR: Expected [IntExpr, StringExpr, BooleanExpr, Id] got '}' on line 3
 PARSER: Parse failed with 1 error(s)

 CST for program 3: Skipped due to PARSER error(s)
```

All three of these programs fail due to Parse errors, and thus do not print out the CSTs. The first program fails because we have an incomplete int expression. The parser found the "=", so it is looking for an expression after and instead found the closing brace.The second program fails because since we do not have a closing brace to the program, the follow of StatementList, the program expects a statement. Instead it finds the EOP, so it throws an error. The third program fails because we have an incomplete assign statement. The program finds the id followed by "=", so it is expecting an expression but instead finds a closing brace.

Test Cases Non-Verbose Mode

Input text:

```
{intaintba=0b=0while(a!=3) {print(a)while(b!=3) {print(b)b=1+bif(b==2){print("there
 is no spoon")}}b=0a=1+a}}$

{while a == 5 {}}$
```

Output:

```
INFO  Lexer - Lexing program 1...
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 1 ...
PARSER: Parse completed successfully

CST for program 1 ...
```

```
<Program>
-<Block>
--[{]
--<StatementList>
---<Statement>
----<VarDecl>
-----<Type>
------[int]
-----<Id>
------[a]
---<StatementList>
----<Statement>
-----<VarDecl>
------<Type>
-------[int]
------<Id>
-------[b]
----<StatementList>
-----<Statement>
------<AssignStatement>
-------<Id>
--------[a]
-------[=]
-------<Expression>
--------<IntegerExpression>
---------<Digit>
----------[0]
-----<StatementList>
------<Statement>
-------<AssignStatement>
--------<Id>
---------[b]
--------[=]
--------<Expression>
---------<IntegerExpression>
----------<Digit>
-----------[0]
------<StatementList>
-------<Statement>
--------<WhileStatement>
---------[while]
---------<BooleanExpression>
----------[(]
----------<Expression>
-----------<Id>
------------[a]
----------<BoolOp>
-----------[!=]
----------<Expression>
-----------<IntegerExpression>
------------<Digit>
-------------[3]
```

```
----------[)]
---------<Block>
----------[{]
----------<StatementList>
-----------<Statement>
------------<PrintStatement>
-------------[print]
-------------[(]
-------------<Expression>
--------------<Id>
---------------[a]
-------------[)]
-----------<StatementList>
------------<Statement>
-------------<WhileStatement>
--------------[while]
--------------<BooleanExpression>
---------------[(]
---------------<Expression>
----------------<Id>
-----------------[b]
---------------<BoolOp>
----------------[!=]
---------------<Expression>
----------------<IntegerExpression>
-----------------<Digit>
------------------[3]
---------------[)]
--------------<Block>
---------------[{]
---------------<StatementList>
----------------<Statement>
-----------------<PrintStatement>
------------------[print]
------------------[(]
------------------<Expression>
-------------------<Id>
--------------------[b]
------------------[)]
----------------<StatementList>
-----------------<Statement>
------------------<AssignStatement>
-------------------<Id>
--------------------[b]
-------------------[=]
-------------------<Expression>
--------------------<IntegerExpression>
---------------------<Digit>
----------------------[1]
---------------------<IntOp>
----------------------[+]
---------------------<Expression>
```

```
----------------------<Id>
-----------------------[b]
-----------------<StatementList>
------------------<Statement>
-------------------<IfStatement>
--------------------[if]
--------------------<BooleanExpression>
---------------------[(]
---------------------<Expression>
----------------------<Id>
-----------------------[b]
---------------------<BoolOp>
----------------------[==]
---------------------<Expression>
----------------------<IntegerExpression>
-----------------------<Digit>
------------------------[2]
---------------------[)]
--------------------<Block>
---------------------[{]
---------------------<StatementList>
----------------------<Statement>
----------------------<PrintStatement>
-----------------------[print]
-----------------------[(]
-----------------------<Expression>
------------------------<StringExpression>
-------------------------["]
-------------------------<CharList>
--------------------------<Char>
--------------------------[t]
--------------------------<CharList>
---------------------------<Char>
---------------------------[h]
---------------------------<CharList>
----------------------------<Char>
----------------------------[e]
----------------------------<CharList>
-----------------------------<Char>
-----------------------------[r]
-----------------------------<CharList>
------------------------------<Char>
------------------------------[e]
------------------------------<CharList>
-------------------------------<Char>
-------------------------------[ ]
-------------------------------<CharList>
--------------------------------<Char>
--------------------------------[i]
--------------------------------<CharList>
---------------------------------<Char>
---------------------------------[s]
```

```
---------------------------------<CharList>
----------------------------------<Char>
-----------------------------------[ ]
----------------------------------<CharList>
-----------------------------------<Char>
------------------------------------[n]
-----------------------------------<CharList>
------------------------------------<Char>
-------------------------------------[o]
------------------------------------<CharList>
-------------------------------------<Char>
--------------------------------------[ ]
-------------------------------------<CharList>
--------------------------------------<Char>
---------------------------------------[s]
--------------------------------------<CharList>
---------------------------------------<Char>
----------------------------------------[p]
---------------------------------------<CharList>
----------------------------------------<Char>
-----------------------------------------[o]
----------------------------------------<CharList>
-----------------------------------------<Char>
------------------------------------------[o]
-----------------------------------------<CharList>
------------------------------------------<Char>
-------------------------------------------[n]
--------------------------["]
------------------------[)]
--------------------[}]
---------------[}]
------------<StatementList>
-------------<Statement>
--------------<AssignStatement>
---------------<Id>
----------------[b]
---------------[=]
---------------<Expression>
----------------<IntegerExpression>
-----------------<Digit>
------------------[0]
-------------<StatementList>
--------------<Statement>
---------------<AssignStatement>
----------------<Id>
-----------------[a]
----------------[=]
----------------<Expression>
-----------------<IntegerExpression>
------------------<Digit>
-------------------[1]
------------------<IntOp>
```

```
------------------[+]
-----------------<Expression>
------------------<Id>
--------------------[a]
----------[}]
--[}]
-[$]


INFO  Lexer - Lexing program 2...
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 2 ...
PARSER: ERROR: Expected [BooleanExpression] got 'a' on line 3
PARSER: Parse failed with 1 error(s)

CST for program 2: Skipped due to PARSER error(s)
```

When running in Non-Verbose mode, the lexical analysis does not print out the token sequence, and the parser doesn't output each stage as it is parsing. The first program does not throw any errors in lex or parse, so it continues to print out the CST. However, an error is thrown in the second program during the parsing because we do not have a BooleanExpression following "while", so it is output and the CST is skipped.

Test Cases Non-Verbose Mode Errors

Input text:

```
{print(3)int a a = 1}{}$
{3 ( "" print } }$
{int b b = 33}$
{int a/*unclosed comment}$
```

Output:

```
INFO  Lexer - Lexing program 1...
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 1 ...
PARSER: ERROR: Expected [EOP] got '{' on line 1
PARSER: Parse failed with 1 error(s)

CST for program 1: Skipped due to PARSER error(s)


INFO  Lexer - Lexing program 2...
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 2 ...
PARSER: ERROR: Expected [PrintStatement, AssignStatement, VarDecl, WhileStatement,
IfStatement, Block] got '3' on line 2
PARSER: Parse failed with 1 error(s)

CST for program 2: Skipped due to PARSER error(s)
```

```
INFO  Lexer - Lexing program 3...
INFO  Lexer - Lex completed with 0 errors

PARSER: Parsing program 3 ...
PARSER: ERROR: Expected [PrintStatement, AssignStatement, VarDecl, WhileStatement,
IfStatement, Block] got '3' on line 3
PARSER: Parse failed with 1 error(s)

CST for program 3: Skipped due to PARSER error(s)


INFO  Lexer - Lexing program 4...
WARNING Lexer - Missing EOP Character '$'
WARNING Lexer - Unclosed Comment at End of Program

PARSER: Parsing program 4 ...
PARSER: ERROR: Expected [StatementList] got 'end of stream' on line 4
PARSER: Parse failed with 1 error(s)

CST for program 4: Skipped due to PARSER error(s)
```

Sine all of these programs throw parse errors, none of them print out the CST, and since we are in non-verbose mode, only the begin statements, completed statements, and error messages will be shown. The first one fails because we close the block brace, so the program is expecting an EOP token to come next, but instead there is another opening brace. The second program fails because after the opening brace, the program expects a statement list, but instead it finds a digit, which in our grammar is not a StatementList. The third program fails because of the second '3' in b = 33. Digits in our grammar can only be a single number, so after the parser finds b = 3, it is expecting a statement, but instead gets '3'. The fourth program fails parse because after finding int a, the parser recognizes this as an assign statement, StatementList :== Statement StatementList, so it is still waiting for the StatementList. The unclosed comment causes the parser to reach the end of stream without finding a StatementList (or nothing, which would result in a closing brace).