Project Two : Parser Output

Jordan Murray

Jordan.Murray1@Marist.edu

April 5, 2021

NOTE: This document excludes the lexer output for conciseness

1 Test Case 1

1.1 Program

{}$

1.2 output

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

CST for program 1
<Program>
-<Block>
–[{]
–[Statement List]
–[}]
-[$]

1.3 comments

This is a simple test program that completes with no errors or warnings

2 Test Case 2

2.1 Program

{{{{{{}}}}}}$

2.2 output

PARSER: Parsing program 2
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmtList()
PARSER: parseStmtList()
PARSER: parseStmtList()
PARSER: parseStmtList()
PARSER: parseStmtList()
Parse completed successfully

CST for program 2
<Program>
-<Block>
–[{]
–<Statement List>
—<Statement>
—-<Block>
——[{]
——<Statement List>
———<Statement>
———-<Block>
————[{]
————<Statement List>
—————<Statement>
—————-<Block>
——————[{]

————<Statement List>
————<Statement>
————-<Block>
————[{]
————<Statement List>
————<Statement>
————-<Block>
————[{]
————[Statement List]
————[}]
————[}]
————[}]
———[}]
——[}]
–[}]
-[$]

2.3 COMMENTS

This program tests the parser to ensure that all of the braces match.

3 Test Case 3

3.1 Program

{{{{{}}} /* comments are ignored */ }}}}$

3.2 OUTPUT

PARSER: Parsing program 3
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseBlock()
PARSER: parseStmtList()

PARSER: parseStmtList()
PARSER: parseStmtList()
PARSER: parseStmtList()
PARSER: parseStmtList()
PARSER: parseStmtList()
PARSER: ERROR: Expected [T_EOP] got [R_BRACE] with value '}' on line 5
Parse completed with 1 error(s)

CST skipped for program 3 due to PARSER error(s)

3.3 comments

This program is similar to number 2, but shows the error that is presented when the curly braces do not match up.

4 Test Case 4

4.1 Program

{/* comments are still ignored */ int@}$

4.2 output

PARSER: skipped for program 4 due to Lexer error(s)

CST skipped for program 4 due to Lexer error(s)

4.3 comments

This program tests when lexer results in an error, hence it will not move on to parse.

5 Test Case 5

5.1 Program

{
int a
a = a
string b
a = b
}$

5.2 output

PARSER: Parsing program 5
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()

PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseAssignStmt()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseExpr()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseAssignStmt()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseExpr()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseStmtList()
Parse completed successfully

CST for program 5
<Program>
-<Block>
–[{]
–<Statement List>
—<Statement>
—-<Variable Declaration>
——<Type>
———[int]
——<Id>
———<Char>
———-[a]
—<Statement List>
—-<Statement>
——<Assignment Statement>
———<Id>
———-<Char>
————[a]
———<=>
———-<Expr>

———<Id>
———<Char>
———-[a]
——<Statement List>
——<Statement>
——-<Variable Declaration>
———<Type>
———[string]
———<Id>
———<Char>
———-[b]
——<Statement List>
——-<Statement>
———<Assignment Statement>
———<Id>
———-<Char>
————[a]
———<=>
———-<Expr>
————<Id>
—————<Char>
—————-[b]
—[}]
–[$]

5.3 COMMENTS

This program tests basic int and string declarations and completes with no errors becuase parse does not do type checking

6 Test Case 6

6.1 Program

{"inta"}$

6.2 OUTPUT

PARSER: Parsing program 6
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: ERROR: invalid statement – Expected [print, assign, while, if, var declaration, or block statement] got [T_QUOTE] with value '"' on line 16
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: ERROR: invalid statement – Expected [print, assign, while, if, var declaration, or block statement

] got [T_CHAR] with value 'i' on line 16
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: ERROR: invalid statement – Expected [print, assign, while, if, var declaration, or block statement] got [T_CHAR] with value 'n' on line 16
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: ERROR: invalid statement – Expected [print, assign, while, if, var declaration, or block statement] got [T_CHAR] with value 't' on line 16
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: ERROR: invalid statement – Expected [print, assign, while, if, var declaration, or block statement] got [T_CHAR] with value 'a' on line 16
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: ERROR: invalid statement – Expected [print, assign, while, if, var declaration, or block statement] got [T_QUOTE] with value '"' on line 16
PARSER: parseStmtList()
ERROR: Parent for Program is null
Parse completed with 6 error(s)

CST skipped for program 6 due to PARSER error(s)

6.3 COMMENTS

This program throws many errors because it does not expect to see a quote in a statement. Once it is in the statment, the recursion loops there looking for a valid statement and it does not find anything, hence why it looks like the same error is being printed multiple times, on different tokens. Since there were parse errors, the CST is not printed.

7 Test Case 7

7.1 Program

/*LongTestCase-EverythingExceptBooleanDeclaration*/{/*IntDeclaration*/intaintba=0b=0/*WhileLoop*/while(a!=3){pri
is no spoon"/*Thiswilldonothing*/)}}b=0a=1+a}}$

7.2 OUTPUT

PARSER: Parsing program 7
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseStmtList()

PARSER: parseStmt()
PARSER: parseVarDecl()
PARSER: parseType()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseAssignStmt()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseDigit()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseAssignStmt()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseDigit()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseWhileStmt()
PARSER: parseBoolExpr()
PARSER: parseExpr()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseBoolOp()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseDigit()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parsePrintStmt()
PARSER: parseExpr()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseWhileStmt()
PARSER: parseBoolExpr()
PARSER: parseExpr()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseBoolOp()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseDigit()
PARSER: parseBlock()
PARSER: parseStmtList()

PARSER: parseStmt()
PARSER: parsePrintStmt()
PARSER: parseExpr()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseAssignStmt()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseDigit()
PARSER: parseIntOp()
PARSER: parseExpr()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseIfStmt()
PARSER: parseBoolExpr()
PARSER: parseExpr()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseBoolOp()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseDigit()
PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parsePrintStmt()
PARSER: parseExpr()
PARSER: parseStringExpr()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()

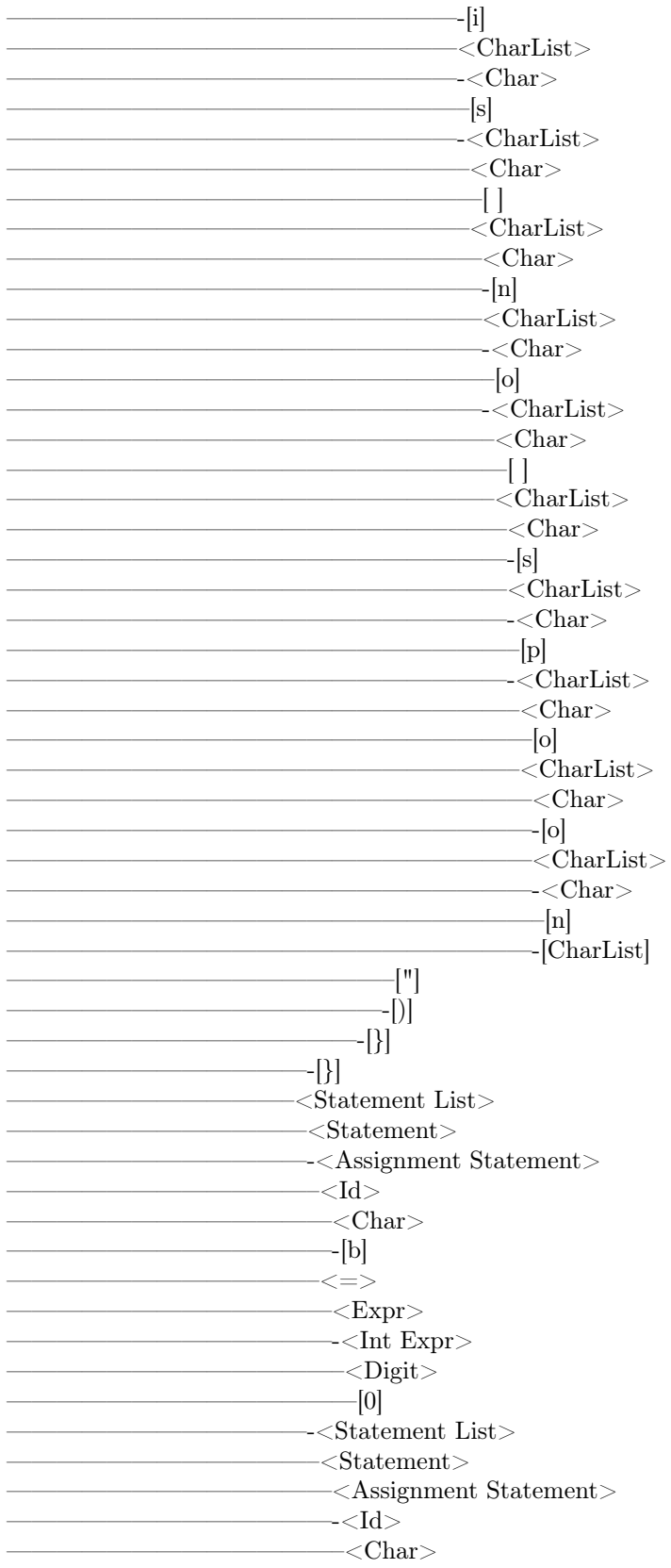
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseChar()
PARSER: parseCharList()
PARSER: parseStmtList()
PARSER: parseStmtList()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseAssignStmt()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseDigit()
PARSER: parseStmtList()
PARSER: parseStmt()
PARSER: parseAssignStmt()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseDigit()
PARSER: parseIntOp()
PARSER: parseExpr()
PARSER: parseId()
PARSER: parseChar()
PARSER: parseStmtList()
PARSER: parseStmtList()
Parse completed successfully

CST for program 7
<Program>
-<Block>
–[{]
–<Statement List>
—<Statement>
—-<Variable Declaration>
——<Type>

——[int]
—<Id>
——<Char>
——-[a]
—<Statement List>
—-<Statement>
——<Variable Declaration>
——<Type>
——-[int]
——<Id>
——-<Char>
———[b]
—-<Statement List>
——<Statement>
——<Assignment Statement>
——-<Id>
———<Char>
————[a]
——-<=>
———<Expr>
————<Int Expr>
————-<Digit>
————[0]
——<Statement List>
——-<Statement>
———<Assignment Statement>
————<Id>
————-<Char>
—————[b]
————<=>
————-<Expr>
—————<Int Expr>
——————<Digit>
——————-[0]
———<Statement List>
————<Statement>
————-<While Statement>
—————<while>
——————<Boolean Expr>
——————-<(>
———————<Expr>
———————<Id>
———————-<Char>
————————[a]
———————<Bool Op>
————————[!=]
———————<Expr>
————————<Int Expr>
————————-<Digit>
————————[3]
———————[)]
———————<Block>

———————[{]
———————<Statement List>
———————-<Statement>
———————<Print Statement>
———————<print>
———————-<(>
———————<Expr>
———————<Id>
———————-<Char>
———————[a]
———————[)]
———————-<Statement List>
———————<Statement>
———————<While Statement>
———————-<while>
———————<Boolean Expr>
———————<(>
———————-<Expr>
———————<Id>
———————<Char>
———————-[b]
———————-<Bool Op>
———————[!=]
———————-<Expr>
———————<Int Expr>
———————<Digit>
———————-[3]
———————-[)]
———————-<Block>
———————[{]
———————<Statement List>
———————<Statement>
———————-<Print Statement>
———————<print>
———————<(>
———————-<Expr>
———————<Id>
———————<Char>
———————-[b]
———————-[)]
———————<Statement List>
———————-<Statement>
———————<Assignment Statement>
———————<Id>
———————-<Char>
———————[b]
———————<=>
———————-<Expr>
———————<Int Expr>
———————<Digit>
———————-[1]
———————<Int Op>

———————————-[+]
———————————<Expr>
———————————-<Id>
———————————<Char>
———————————[b]
———————————<Statement List>
———————————<Statement>
———————————-<If Statement>
———————————<if>
———————————<Boolean Expr>
———————————-<(>
———————————<Expr>
———————————<Id>
———————————-<Char>
———————————[b]
———————————<Bool Op>
———————————[==]
———————————<Expr>
———————————<Int Expr>
———————————-<Digit>
———————————[2]
———————————[)]
———————————<Block>
———————————[{]
———————————<Statement List>
———————————-<Statement>
———————————<Print Statement>
———————————<print>
———————————-<(>
———————————<Expr>
———————————<String Expr>
———————————-<">
———————————<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]
-[CharList]
["]
-[)]
-[}]
-[}]
<Statement List>
<Statement>
-<Assignment Statement>
<Id>
<Char>
-[b]
<=>
<Expr>
-<Int Expr>
<Digit>
[0]
-<Statement List>
<Statement>
<Assignment Statement>
-<Id>
<Char>

——————————————[a]
——————————————-<=>
——————————————<Expr>
——————————————<Int Expr>
——————————————-<Digit>
——————————————[1]
——————————————-<Int Op>
——————————————[+]
——————————————-<Expr>
——————————————<Id>
——————————————<Char>
——————————————-[a]
————————————-[}]
————————————[}]
————————————-[$]

7.3 COMMENTS

This is the lex without spaces example.

8 Test Case 8

8.1 Program

{/* comments are still ignored */ int@}$

8.2 OUTPUT

PARSER: skipped for program 8 due to Lexer error(s)

CST skipped for program 8 due to Lexer error(s)

8.3 COMMENTS

The @ token is not apart of the grammar, therefore an error is thrown and does not move on to parse

9 Test Case 9

9.1 Program

{/*bad comment } $

9.2 OUTPUT

PARSER: Parsing program 9
PARSER: parse()
PARSER: parseProgram()

PARSER: parseBlock()
PARSER: parseStmtList()
PARSER: ERROR: Expected [T_EOP] got [L_BRACE] with value '{' on line 22
Parse completed with 1 error(s)

CST skipped for program 9 due to PARSER error(s)

9.3 COMMENTS

This program shows the warning thrown if a comment is not closed in lex, but since not an error, move on to parse, which throws the error because it does not find the end of program symbol due to the comment. The EOP symbol does not get added to token stream, so the parser does not pass in parseProgram() when trying to match that token.

10 TEST CASE 10

10.1 PROGRAM

{}

10.2 OUTPUT

INFO LEXER —> Lexing Program 10
DEBUG LEXER —> L_BRACE [{] Found at (24 : 1)
DEBUG LEXER —> R_BRACE [}] Found at (24 : 2)
Warning: Missing End of Program char: $
Adding EOP Token...
DEBUG LEXER —> T_EOP [$] Found at (24 : 0)

PARSER: Parsing program 10
PARSER: parse()
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStmtList()
Parse completed successfully

CST for program 10
<Program>
-<Block>
–[{]
–[Statement List]
–[}]
-[$]

10.3 COMMENTS

The final test case shows the warning thrown in lex if no EOP is found at the end of a file, I included the Lexer output here because I chose to add the EOP back in which is shown above and then send the program to the parser with the EOP added to the token stream, since this is only a warning. This then completes with no errors, just like test case 1.