Project 4

Test Case #1:

Input:

```
@{}
$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
ERROR Lexer - Error: line 1 Unrecognized Token: @
Please reference grammar guide.
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 1
DEBUG Lexer - EOP [ $ ] found on line 2
LEXER: Lex completed with 1 error(s)

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

Test Case #2:

Input:

```
{ x = 1 }
```

Compile Output:

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - ID [ x ] found on line 1
DEBUG Lexer - ASSIGN_OP [ = ] found on line 1
DEBUG Lexer - DIGIT [ 1 ] found on line 1
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 1
ERROR Lexer - Error: last line of program does not
end with "$".
Error Lexer - Lex failed with 1 error(s)

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

Test Case #3:

Input:

```
{ print("foo }
$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - PRINT [ print ] found on line 1
DEBUG Lexer - LPAREN [ ( ] found on line 1
DEBUG Lexer - StringExpr [ start " ] found on line
1
DEBUG Lexer - char [ f ] found on line 1
DEBUG Lexer - char [ o ] found on line 1
DEBUG Lexer - char [ o ] found on line 1
DEBUG Lexer - char [   ] found on line 1
ERROR Lexer - Error: line 1 Unrecognized Token in
string: } Only lowercase letters a through z and
spaces are allowed in strings
Error Lexer - Lex failed with 1 error(s)

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

Test Case #4:

Input:

```
{ print("Hi!"); }
$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - PRINT [ print ] found on line 1
DEBUG Lexer - LPAREN [ ( ] found on line 1
DEBUG Lexer - StringExpr [ start " ] found on line
1
ERROR Lexer - Error: line 1 Unrecognized Token in
string: H Only lowercase letters a through z and
spaces are allowed in strings
Error Lexer - Lex failed with 1 error(s)

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

Test Case #5:

Input:

```
/* */{ }$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
WARNING Lexer - Warning: line 2 - Empty comment
block detected.

DEBUG Lexer - OPEN_BLOCK [ { ] found on line 2
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 2
DEBUG Lexer - EOP [ $ ] found on line 2
LEXER: Lex completed with 0 error(s)

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

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

AST for program 1:
[ BLOCK ]

Program 1 Semantic Analysis produced
0 error(s) and 0 warning(s)

Program 1 Symbol Table
---
Name Type    Scope Line
---

Generated 6502a Machine Code:
00
```

Test Case #6:

Input:

```
/* oops { }$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
ERROR Lexer - Error: Unterminated comment starting
on line 1. Lexing terminated.
Error Lexer - Lex failed with 1 error(s)

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

Test Case #7:

Input:

```
{ x = 1 2 }$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - ID [ x ] found on line 1
DEBUG Lexer - ASSIGN_OP [ = ] found on line 1
DEBUG Lexer - DIGIT [ 1 ] found on line 1
DEBUG Lexer - DIGIT [ 2 ] found on line 1
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 1
DEBUG Lexer - EOP [ $ ] found on line 1
LEXER: Lex completed with 0 error(s)

PARSER: Parsing program 1...
PARSER: parse() called
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignmentStatement()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseStatementList()
PARSER: parseStatement()

PARSER: Parse failed with 1 error

PARSER ERROR: PARSER ERROR: Unexpected token 2 at
line 1 in parseStatement
CST for program 1: Skipped due to PARSER error(s).
```

Test Case #8:

Input:

```
{ x = 1 $
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - ID [ x ] found on line 1
DEBUG Lexer - ASSIGN_OP [ = ] found on line 1
DEBUG Lexer - DIGIT [ 1 ] found on line 1
DEBUG Lexer - EOP [ $ ] found on line 1
LEXER: Lex completed with 0 error(s)

PARSER: Parsing program 1...
PARSER: parse() called
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignmentStatement()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseStatementList()

PARSER: Parse failed with 1 error

PARSER ERROR: PARSER ERROR: Expected RBRACE but got
$ at line 1
CST for program 1: Skipped due to PARSER error(s).
```

Test Case #9:

Input:

```
{ 5 = x }$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - DIGIT [ 5 ] found on line 1
DEBUG Lexer - ASSIGN_OP [ = ] found on line 1
DEBUG Lexer - ID [ x ] found on line 1
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 1
DEBUG Lexer - EOP [ $ ] found on line 1
LEXER: Lex completed with 0 error(s)

PARSER: Parsing program 1...
PARSER: parse() called
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()

PARSER: Parse failed with 1 error

PARSER ERROR: PARSER ERROR: Unexpected token 5 at
line 1 in parseStatement
CST for program 1: Skipped due to PARSER error(s).
```

Test Case #10:

Input:

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

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - IFSTATEMENT [ if ] found on line 1
DEBUG Lexer - LPAREN [ ( ] found on line 1
DEBUG Lexer - ID [ x ] found on line 1
DEBUG Lexer - RPAREN [ ) ] found on line 1
DEBUG Lexer - ID [ x ] found on line 1
DEBUG Lexer - ASSIGN_OP [ = ] found on line 1
DEBUG Lexer - DIGIT [ 3 ] found on line 1
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 1
DEBUG Lexer - EOP [ $ ] found on line 1
LEXER: Lex completed with 0 error(s)

PARSER: Parsing program 1...
PARSER: parse() called
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseIfStatement()
PARSER: parseBooleanExpr()
PARSER: parseExpr()
PARSER: parseBoolOp()

PARSER: Parse failed with 1 error

PARSER ERROR: PARSER ERROR: Expected boolean
operator at line 1
CST for program 1: Skipped due to PARSER error(s).
```

Test Case #11:

Input:

```
{ x = 3; }$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - ID [ x ] found on line 1
DEBUG Lexer - ASSIGN_OP [ = ] found on line 1
DEBUG Lexer - DIGIT [ 3 ] found on line 1
ERROR Lexer - Error: line 1 Unrecognized Token: ;
Please reference grammar guide.
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 1
DEBUG Lexer - EOP [ $ ] found on line 1
LEXER: Lex completed with 1 error(s)

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

Test Case #12:

Input:

```
{ int x; int x; }$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - ITYPE [ int ] found on line 1
DEBUG Lexer - ID [ x ] found on line 1
ERROR Lexer - Error: line 1 Unrecognized Token: ;
Please reference grammar guide.
DEBUG Lexer - ITYPE [ int ] found on line 1
DEBUG Lexer - ID [ x ] found on line 1
ERROR Lexer - Error: line 1 Unrecognized Token: ;
Please reference grammar guide.
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 1
DEBUG Lexer - EOP [ $ ] found on line 1
LEXER: Lex completed with 2 error(s)

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

Test Case #13:

Input:

```
{ int x; x = true; }$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - ITYPE [ int ] found on line 1
DEBUG Lexer - ID [ x ] found on line 1
ERROR Lexer - Error: line 1 Unrecognized Token: ;
Please reference grammar guide.
DEBUG Lexer - ID [ x ] found on line 1
DEBUG Lexer - ASSIGN_OP [ = ] found on line 1
DEBUG Lexer - BOOLVALT [ true ] found on line 1
ERROR Lexer - Error: line 1 Unrecognized Token: ;
Please reference grammar guide.
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 1
DEBUG Lexer - EOP [ $ ] found on line 1
LEXER: Lex completed with 2 error(s)

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

Test Case #14:

Input:

```
{int a a=5 print(a)}$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - ITYPE [ int ] found on line 1
DEBUG Lexer - ID [ a ] found on line 1
DEBUG Lexer - ID [ a ] found on line 1
DEBUG Lexer - ASSIGN_OP [ = ] found on line 1
DEBUG Lexer - DIGIT [ 5 ] found on line 1
DEBUG Lexer - PRINT [ print ] found on line 1
DEBUG Lexer - LPAREN [ ( ] found on line 1
DEBUG Lexer - ID [ a ] found on line 1
DEBUG Lexer - RPAREN [ ) ] found on line 1
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 1
DEBUG Lexer - EOP [ $ ] found on line 1
LEXER: Lex completed with 0 error(s)

PARSER: Parsing program 1...
PARSER: parse() called
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignmentStatement()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parsePrintStatement()
PARSER: parseExpr()
PARSER: parseStatementList()
PARSER: Parse completed successfully

CST for program 1:
<Program>
-<Block>
--[{]
--<Statement List>
---<Statement>
----<VarDecl>
-----<int>
-----<a>
---<Statement>
----<AssignmentStatement>
-----<a>
-----<=>
-----<Expr>
------<IntExpr>
-------<5>
---<Statement>
----<PrintStatement>
-----<print>
-----<(>
-----<Expr>
------<a>
-----<)>
--[}]
-[$]

AST for program 1:
< BLOCK >
-< Statement >
--< Variable Declaration >
---[ int ]
---[ a ]
--< Statement >
---< Assignment Statement >
----[ a ]
----[ 5 ]
--< Statement >
---< Print Statement >
----[ a ]

Program 1 Semantic Analysis produced
0 error(s) and 0 warning(s)

Program 1 Symbol Table
---
Name Type    Scope Line
---
a    int    0    12

Generated 6502a Machine Code:
A9 05 8D 10 00 AC 10 00
A2 01 FF 00
```

```
>
>load
Process ID: 0
>run 0
>5
```

Log

Sun, May 4th 2025, 8:27:57 pm 380
OS Idle
Sun, May 4th 2025, 8:27:46 pm 168
OS Handling IRQ~2

Processes Round Robin

PID	PC	IR	ACC	X	Y	Z	Priority	State	Location

No programs are in execution

Memory

0x008	00	00	00	00	00	00	00	00
0x010	00	00	00	00	00	00	00	00
0x018	00	00	00	00	00	00	00	00
0x020	00	00	00	00	00	00	00	00

Hard Drive

0:0:0	1	000	3030313130300000000000000000000000000000000000000000000000000000
0:0:1	0	000	0000000000000000000000000000000000000000000000000000000000000000

CPU No Instruction

PC	IR	ACC	X	Y	Z
000	0	0	0	0	0

A9 05 8D 10 00 AC
10 00
A2 01 FF 00

Test Case #15:

```
{
  int x
  x = 7
  print(x)
}$
```

Compile

Output:

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 1
DEBUG Lexer - ITYPE [ int ] found on line 2
DEBUG Lexer - ID [ x ] found on line 2
DEBUG Lexer - ID [ x ] found on line 3
DEBUG Lexer - ASSIGN_OP [ = ] found on line 3
DEBUG Lexer - DIGIT [ 7 ] found on line 3
DEBUG Lexer - PRINT [ print ] found on line 4
DEBUG Lexer - LPAREN [ ( ] found on line 4
DEBUG Lexer - ID [ x ] found on line 4
DEBUG Lexer - RPAREN [ ) ] found on line 4
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 5
DEBUG Lexer - EOP [ $ ] found on line 5
LEXER: Lex completed with 0 error(s)

PARSER: Parsing program 1...
PARSER: parse() called
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignmentStatement()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parsePrintStatement()
PARSER: parseExpr()
PARSER: parseStatementList()
PARSER: Parse completed successfully

CST for program 1:
<Program>
-<Block>
--[{]
--<Statement List>
---<Statement>
----<VarDecl>
-----<int>
-----<x>
---<Statement>
----<AssignmentStatement>
-----<x>
-----<=>
-----<Expr>
```

```
------<IntExpr>
-------<7>
---<Statement>
----<PrintStatement>
-----<print>
-----<(>
-----<Expr>
------<x>
-----<)>
--[}]
-[$]

AST for program 1:
< BLOCK >
-< Statement >
--< Variable Declaration >
---[ int ]
---[ x ]
--< Statement >
---< Assignment Statement >
----[ x ]
----[ 7 ]
--< Statement >
---< Print Statement >
----[ x ]

Program 1 Semantic Analysis produced
0 error(s) and 0 warning(s)

Program 1 Symbol Table
---
Name Type    Scope Line
---
x    int    0    2

Generated 6502a Machine Code:
A9 07 8D 10 00 AC 10 00
A2 01 FF 00
```

```
>load
Process ID: 1
>run 1
>7
```

Sun, May 4th 2025, 8:34:08 pm 752

OS **Handling IRQ~2**

Processes Round Robin

PID	PC	IR	ACC	X	Y	Z	Priority	State	Location

No programs are in execution

Memory

0x008	00	00	00	00	00	00	00	00
0x010	00	00	00	00	00	00	00	00
0x018	00	00	00	00	00	00	00	00
0x020	00	00	00	00	00	00	00	00

Hard Drive

0:0:0	1	000	3030313130300000000000000000000000000000000000000000000000000000
0:0:1	0	000	0000000000000000000000000000000000000000000000000000000000000000

CPU No Instruction

PC	IR	ACC	X	Y	Z
000	0	0	0	0	0

A9 07 8D 10 00 AC
10 00
A2 01 FF 00

Test Case #16:

Input:

```
{
  string s
  s = "hello world"
  print(s)
}$
```

```
---------------<o>
---------------<CharList>
----------------<r>
----------------<CharList>
-----------------<l>
-----------------<CharList>
------------------<d>
------------------<CharList>
-------------------<Îµ>
-------<">
---<Statement>
----<PrintStatement>
-----<print>
-----<(>
-----<Expr>
------<s>
-----<)>
--[}]
-[$]

AST for program 1:
< BLOCK >
-< Statement >
--< Variable Declaration >
---[ string ]
---[ s ]
--< Statement >
---< Assignment Statement >
----[ s ]
----[ hello world ]
--< Statement >
---< Print Statement >
----[ s ]

Program 1 Semantic Analysis produced
0 error(s) and 0 warning(s)

Program 1 Symbol Table
---
Name Type    Scope Line
---
s    string 0    2

Generated 6502a Machine Code:
A0 11 A2 02 FF 8D 10 00
AC 10 00 A2 01 FF 00 00
00 68 65 6C 6C 6F 20 77
6F 72 6C 64 00
```

```
>load
Process ID: 0
>run 0
>hello world0
```

Log

Mon, May 5th 2025, 9:10:40 pm 650

OS Idle

Mon, May 5th 2025, 9:10:16 pm 167

OS Handling IRQ~2

Processes Round Robi

PID	PC	IR	ACC	X	Y	Z	Priority	State	Locati

No programs are in execution

Memory

0x008	00	00	00	00	00	00	00	00
0x010	00	00	00	00	00	00	00	00
0x018	00	00	00	00	00	00	00	00
0x020	00	00	00	00	00	00	00	00

Hard Drive

```
0:0:0  1  000  3030313130300000000000000000000000000000000000000000000000000000
0:0:1  0  000  0000000000000000000000000000000000000000000000000000000000000000
```

CPU No Instruction

PC	IR	ACC	X	Y	Z
000	0	0	0	0	0

A0 11 A2 02 FF
8D 10 00
AC 10 00 A2 01
FF 00 00
00 68 65 6C 6C

Test Case #17:

Input:

```
{
  string s
  s = "hello world"
if ( s == 2){
 print(s)
}}$
```

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 10
DEBUG Lexer - ITYPE [ string ] found on line 11
DEBUG Lexer - ID [ s ] found on line 11
DEBUG Lexer - ID [ s ] found on line 12
DEBUG Lexer - ASSIGN_OP [ = ] found on line 12
DEBUG Lexer - StringExpr [ start " ] found on line
12
DEBUG Lexer - char [ h ] found on line 12
DEBUG Lexer - char [ e ] found on line 12
DEBUG Lexer - char [ l ] found on line 12
DEBUG Lexer - char [ l ] found on line 12
DEBUG Lexer - char [ o ] found on line 12
DEBUG Lexer - char [   ] found on line 12
DEBUG Lexer - char [ w ] found on line 12
DEBUG Lexer - char [ o ] found on line 12
DEBUG Lexer - char [ r ] found on line 12
DEBUG Lexer - char [ l ] found on line 12
DEBUG Lexer - char [ d ] found on line 12
DEBUG Lexer - StringExpr [ end " ] found on line
12
DEBUG Lexer - IFSTATEMENT [ if ] found on line 13
DEBUG Lexer - LPAREN [ ( ] found on line 13
DEBUG Lexer - ID [ s ] found on line 13
DEBUG Lexer - BOOL_EQUAL [ == ] found on line 13
DEBUG Lexer - DIGIT [ 2 ] found on line 13
DEBUG Lexer - RPAREN [ ) ] found on line 13
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 13
DEBUG Lexer - PRINT [ print ] found on line 14
DEBUG Lexer - LPAREN [ ( ] found on line 14
DEBUG Lexer - ID [ s ] found on line 14
DEBUG Lexer - RPAREN [ ) ] found on line 14
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 15
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 15
DEBUG Lexer - EOP [ $ ] found on line 15
LEXER: Lex completed with 0 error(s)

PARSER: Parsing program 1...
PARSER: parse() called
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseStatementList()
PARSER: parseStatement()
```

```
PARSER: parseAssignmentStatement()
PARSER: parseExpr()
PARSER: parseStringExpr()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseIfStatement()
PARSER: parseBooleanExpr()
PARSER: parseExpr()
PARSER: parseBoolOp()
PARSER: parseExpr()
PARSER: parseIntExpr()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parsePrintStatement()
PARSER: parseExpr()
PARSER: parseStatementList()
PARSER: parseStatementList()
PARSER: Parse completed successfully

CST for program 1:
<Program>
-<Block>
--[{]
--<Statement List>
---<Statement>
----<VarDecl>
-----<string>
-----<s>
---<Statement>
----<AssignmentStatement>
-----<s>
-----<=>
-----<Expr>
------<StringExpr>
```

```
-------<">
-------<CharList>
--------<h>
--------<CharList>
---------<e>
---------<CharList>
----------<l>
----------<CharList>
-----------<l>
-----------<CharList>
------------<o>
------------<CharList>
-------------< >
-------------<CharList>
--------------<w>
--------------<CharList>
---------------<o>
---------------<CharList>
----------------<r>
----------------<CharList>
-----------------<l>
-----------------<CharList>
------------------<d>
------------------<CharList>
-------------------<Îµ>
-------<">
---<Statement>
----<IfStatement>
-----<if>
-----<BooleanExpr>
------<(>
------<Expr>
-------<s>
------<BoolOp>
-------<==>
------<Expr>
-------<IntExpr>
--------<2>
------<)>
-----<Block>
------[{]
------<Statement List>
-------<Statement>
--------<PrintStatement>
---------<print>
---------<(>
---------<Expr>
----------<s>
```

```
---------<)>
------[}]
--[}]
-[$]

AST for program 1:
< BLOCK >
-< Statement >
--< Variable Declaration >
---[ string ]
---[ s ]
--< Statement >
---< Assignment Statement >
----[ s ]
----[ hello world ]
--< Statement >
---< If Statement >
----[ if ]
----< BooleanExpr >
-----< Expr >
------[ s ]
-----< Expr >
------< IntExpr >
-------[ 2 ]
----< BLOCK >
-----< Statement >
------< Print Statement >
-------[ s ]

Program 1 Semantic Analysis produced
0 error(s) and 0 warning(s)

Program 1 Symbol Table
---
Name Type    Scope Line
---
s    string 0    11

Generated 6502a Machine Code:
A0 11 A2 02 FF 8D 10 00
AD 10 00 C9 00 F0 00 00
00 68 65 6C 6C 6F 20 77
6F 72 6C 64 00
```

Test Case #18:

Input:

```
{
  string s
  s = "hello world"
while( s == 3){
 print(s)
}}$
```

Compile

Output:

```
DEBUG: Running in verbose mode

LEXER - Lexing program 1...
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 12
DEBUG Lexer - ITYPE [ string ] found on line 13
DEBUG Lexer - ID [ s ] found on line 13
DEBUG Lexer - ID [ s ] found on line 14
DEBUG Lexer - ASSIGN_OP [ = ] found on line 14
DEBUG Lexer - StringExpr [ start " ] found on line 14
DEBUG Lexer - char [ h ] found on line 14
DEBUG Lexer - char [ e ] found on line 14
DEBUG Lexer - char [ l ] found on line 14
DEBUG Lexer - char [ l ] found on line 14
DEBUG Lexer - char [ o ] found on line 14
DEBUG Lexer - char [   ] found on line 14
DEBUG Lexer - char [ w ] found on line 14
DEBUG Lexer - char [ o ] found on line 14
DEBUG Lexer - char [ r ] found on line 14
DEBUG Lexer - char [ l ] found on line 14
DEBUG Lexer - char [ d ] found on line 14
DEBUG Lexer - StringExpr [ end " ] found on line 14
DEBUG Lexer - WHILE [ while ] found on line 15
DEBUG Lexer - LPAREN [ ( ] found on line 15
DEBUG Lexer - ID [ s ] found on line 15
DEBUG Lexer - BOOL_EQUAL [ == ] found on line 15
DEBUG Lexer - DIGIT [ 3 ] found on line 15
DEBUG Lexer - RPAREN [ ) ] found on line 15
DEBUG Lexer - OPEN_BLOCK [ { ] found on line 15
DEBUG Lexer - PRINT [ print ] found on line 16
DEBUG Lexer - LPAREN [ ( ] found on line 16
DEBUG Lexer - ID [ s ] found on line 16
DEBUG Lexer - RPAREN [ ) ] found on line 16
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 17
DEBUG Lexer - CLOSE_BLOCK [ } ] found on line 17
DEBUG Lexer - EOP [ $ ] found on line 17
LEXER: Lex completed with 0 error(s)

PARSER: Parsing program 1...
PARSER: parse() called
PARSER: parseProgram()
PARSER: parseBlock()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseVarDecl()
PARSER: parseStatementList()
PARSER: parseStatement()
PARSER: parseAssignmentStatement()
PARSER: parseExpr()
PARSER: parseStringExpr()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
PARSER: parseCharList()
```

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

CST for program 1:
<Program>
-<Block>
--[{]
--<Statement List>
---<Statement>
----<VarDecl>
-----<string>
-----<s>
---<Statement>
----<AssignmentStatement>
-----<s>
-----<=>
-----<Expr>
------<StringExpr>
-------<">
-------<CharList>
--------<h>
--------<CharList>
---------<e>
---------<CharList>
----------<l>
----------<CharList>
-----------<l>
-----------<CharList>
------------<o>
------------<CharList>
-------------< >
-------------<CharList>
--------------<w>
--------------<CharList>
---------------<o>
---------------<CharList>
```

```
------[{]
------<Statement List>
-------<Statement>
--------<PrintStatement>
---------<print>
---------<(>
---------<Expr>
----------<s>
---------<)>
------[}]
--[}]
-[$]

AST for program 1:
< BLOCK >
-< Statement >
--< Variable Declaration >
---[ string ]
---[ s ]
--< Statement >
---< Assignment Statement >
----[ s ]
----[ hello world ]
--< Statement >
---< While Statement >
----[ while ]
----< BooleanExpr >
-----< Expr >
------[ s ]
-----< Expr >
------< IntExpr >
-------[ 3 ]
----< BLOCK >
-----< Statement >
------< Print Statement >
-------[ s ]

Program 1 Semantic Analysis produced
0 error(s) and 0 warning(s)

Program 1 Symbol Table
---
Name Type    Scope Line
---
s    string 0    13

Generated 6502a Machine Code:
A0 11 A2 02 FF 8D 10 00
A2 00 EC 10 00 C0 01 D0
03 4C 08 00 00 68 65 6C
6C 6F 20 77 6F 72 6C 64
00
```

Test Case #19:

```
{
  string s
  s = "hello"
print(s)
}$
```

ut:

Compile

Output:

```
--------<CharList>
---------<e>
---------<CharList>
----------<l>
----------<CharList>
-----------<l>
-----------<CharList>
------------<o>
------------<CharList>
-------------<Îµ>
-------<">
---<Statement>
----<PrintStatement>
-----<print>
-----<(>
-----<Expr>
------<s>
-----<)>
--[}]
-[$]

AST for program 1:
< BLOCK >
-< Statement >
--< Variable Declaration >
---[ string ]
---[ s ]
--< Statement >
---< Assignment Statement >
----[ s ]
----[ hello ]
--< Statement >
---< Print Statement >
----[ s ]

Program 1 Semantic Analysis produced
0 error(s) and 0 warning(s)

Program 1 Symbol Table
---
Name Type    Scope Line
---
s    string 0    3

Generated 6502a Machine Code:
A0 11 A2 02 FF 8D 10 00
AC 10 00 A2 01 FF 00 00
00 68 65 6C 6C 6F 00
```

```
Process ID: 12
>run 12
>hello0
```

0x010	00	00	00	00	00	00	00	00
0x018	00	00	00	00	00	00	00	00
0x020	00	00	00	00	00	00	00	00

Hard Drive

```
0:0:0  1  000  3030313130300000000000000000000000000000000000000000000000000000
0:0:1  0  000  0000000000000000000000000000000000000000000000000000000000000000
```

CPU No Instruction

PC	IR	ACC	X	Y	Z
000	0	0	0	0	0

```
8D 10 00
AC 10 00 A2 01
FF 00 00
00 68 65 6C 6C
6F 00
```

AI Reflection:

ChatGPT was extremely helpful in generating test cases as always since it's harder for me to think of tons of different unique ones. By having an outside perspective on what

should be tested it will most likely cover things that I wouldn't have thought of while writing the code. AI was also very helpful in finding bugs and helping me spot duplicate code out of the approximately 1500 lines of code that would have otherwise been a nightmare. I think that ChatGPT is getting smarter as new versions are coming out and it learns more from its users, but it still tends to really get stuck up on some issues and not actually provide helpful changes that I could make. It seemed to also have a better memory of conversations we have had and pulling from information previously provided to it, but it still would disregard very important aspects such as our language grammar that I was trying my best to adhere to and it kept fighting me on that.