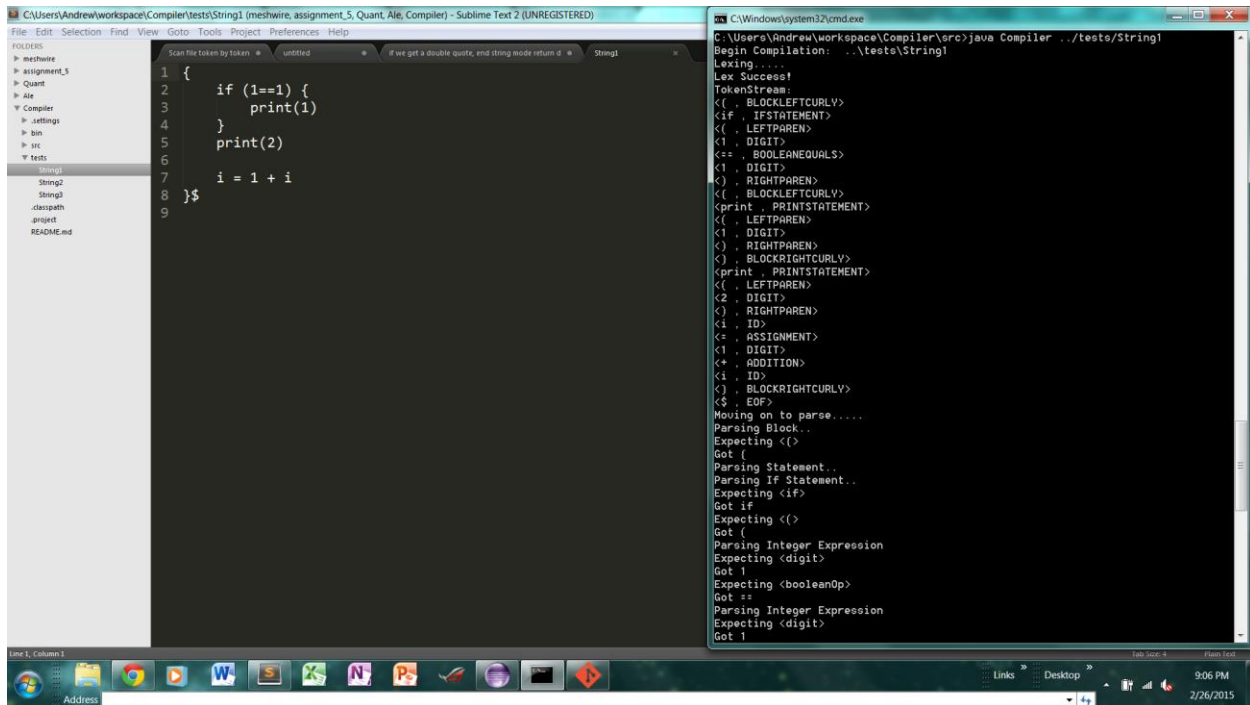


Andrew Langford

Design of Compilers

Project 1

Tests/String1

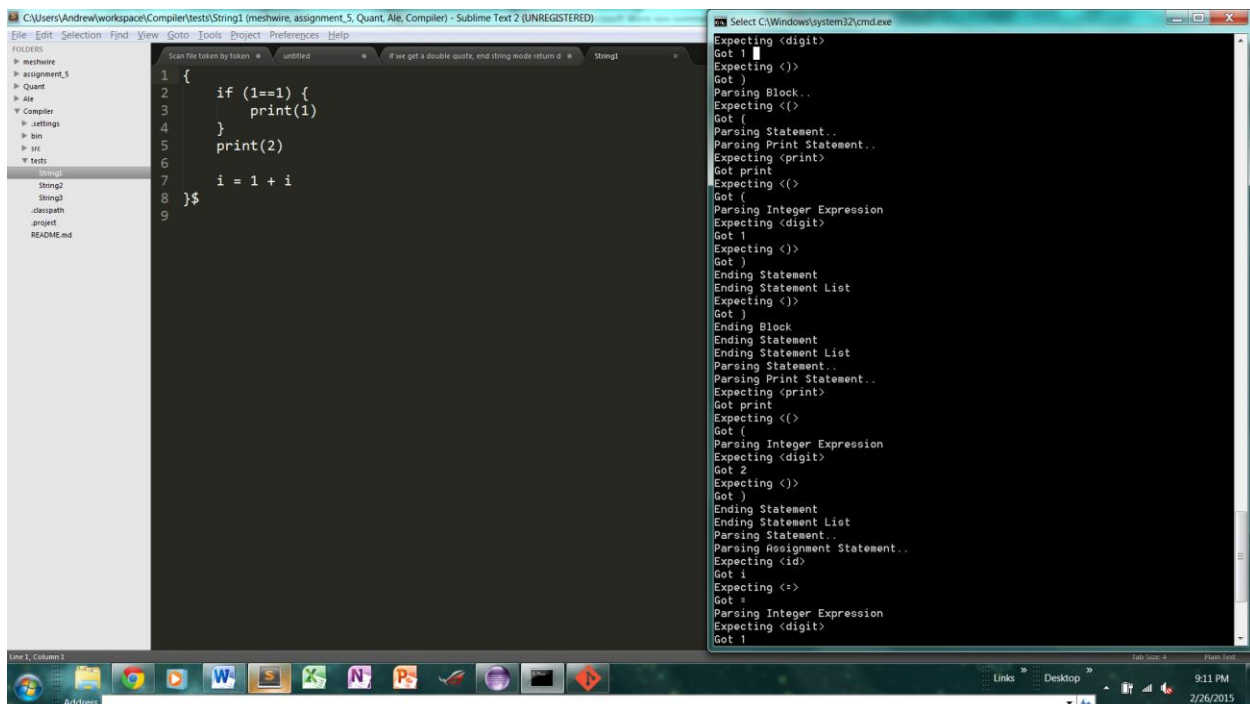


The screenshot shows a Sublime Text editor window on the left with a Java file named `String1.java`. The code is as follows:

```
1 {
2     if (1==1) {
3         print(1)
4     }
5     print(2)
6
7     i = 1 + i
8 }$
9
```

On the right, a command prompt window shows the execution of the Java compiler:

```
C:\Users\Andrew\workspace\Compiler>java Compiler ../tests/String1
Begin Compilation: ..\tests\String1
Lexing.....
Lex Success!
TokenStream:
<(< , BLOCKLEFTCURLY>
<(if , IFSTATEMENT>
<(< , LEFTPAREN>
<1 , DIGIT>
<== , BOOLEANEQUALS>
<1 , DIGIT>
<,> , RIGHTPAREN>
<(< , BLOCKLEFTCURLY>
<print , PRINTSTATEMENT>
<(< , LEFTPAREN>
<1 , DIGIT>
<,> , RIGHTPAREN>
<,> , BLOCKRIGHTCURLY>
<print , PRINTSTATEMENT>
<(< , LEFTPAREN>
<2 , DIGIT>
<,> , RIGHTPAREN>
<1 , ID>
<= , ASSIGNMENT>
<1 , DIGIT>
<+ , ADDITION>
<1 , ID>
<,> , BLOCKRIGHTCURLY>
<$ , EOF>
Moving on to parse.....
Parsing Block..
Expecting <(>
Got (
Parsing Statement..
Parsing If Statement..
Expecting <(if>
Got if
Expecting <(>
Got (
Parsing Integer Expression
Expecting <digit>
Got 1
Expecting <booleanOp>
Got ==
Parsing Integer Expression
Expecting <digit>
Got 1
```



The screenshot shows the same Sublime Text editor window on the left. The command prompt window on the right continues the compilation process:

```
Expecting <digit>
Got 1
Expecting <(>
Got )
Parsing Block..
Expecting <(>
Got (
Parsing Statement..
Parsing Print Statement..
Expecting <print>
Got print
Expecting <(>
Got (
Parsing Integer Expression
Expecting <digit>
Got 1
Expecting <(>
Got )
Ending Statement
Ending Statement List
Expecting <(>
Got )
Ending Block
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Print Statement..
Expecting <print>
Got print
Expecting <(>
Got (
Parsing Integer Expression
Expecting <digit>
Got 2
Expecting <(>
Got )
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Assignment Statement..
Expecting <id>
Got i
Expecting <=>
Got =
Parsing Integer Expression
Expecting <digit>
Got 1
```

The screenshot shows a Sublime Text 2 window with a C program in the main editor and its compilation output in a separate window.

String1.c:

```
1 {
2     if (1==1) {
3         print(1)
4     }
5     print(2)
6
7     i = 1 + i
8 }$
9
```

Compilation Output (C:\Windows\system32\cmd.exe):

```
Got 1
Expecting <(>
Got )
Ending Statement
Ending Statement List
Expecting <(>
Got )
Ending Block
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Print Statement..
Expecting <print>
Got print
Expecting <(>
Got (
Parsing Integer Expression
Expecting <digit>
Got 2
Expecting <(>
Got )
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Assignment Statement..
Expecting <id>
Got i
Expecting <+>
Got +
Parsing Integer Expression
Expecting <digit>
Got 1
Expecting <+>
Got +
Expecting <id>
Got i
Ending Statement
Ending Statement List
Expecting <(>
Got )
Ending Block
Expecting <$>
Got $
Ending Program
Parse Success!
```

Tests/String2

The screenshot shows a Sublime Text 2 window with a C program in the main editor and its compilation output in a separate window.

String2.c:

```
1 {
2     int i
3     i = 0
4
5     int j
6     j = 0
7
8     while (j == 0) {
9         i = 1 + i
10
11         if (i == 3) {
12             j = 1
13         }
14
15         print(1)
16         print(i)
17         print(2)
18         print(j)
19         print(3)
20     }
21
22     print(4)
23 }
24
```

Compilation Output (C:\Windows\system32\cmd.exe):

```
Begin Compilation: ..\tests\String2
Lexing....
Lex Success!
TokenStream:
<( , BLOCKLEFTCURLV>
<int , INTTYPE>
<i , ID>
<= , ASSIGNMENT>
<0 , DIGIT>
<int , INTTYPE>
<j , ID>
<= , ASSIGNMENT>
<0 , DIGIT>
<while , WHILELOOP>
<( , LEFTPAREN>
<j , ID>
<= , BOOLEANEQUALS>
<0 , DIGIT>
< , RIGHTPAREN>
<( , BLOCKLEFTCURLV>
<i , ID>
<= , ASSIGNMENT>
<1 , DIGIT>
<+ , ADDITION>
<i , ID>
<if , IFSTATEMENT>
<( , LEFTPAREN>
<i , ID>
<= , BOOLEANEQUALS>
<3 , DIGIT>
< , RIGHTPAREN>
<( , BLOCKLEFTCURLV>
<j , ID>
<= , ASSIGNMENT>
<1 , DIGIT>
< , BLOCKRIGHTCURLV>
<print , PRINTSTATEMENT>
<( , LEFTPAREN>
<1 , DIGIT>
< , RIGHTPAREN>
<print , PRINTSTATEMENT>
<( , LEFTPAREN>
<i , ID>
< , RIGHTPAREN>
<print , PRINTSTATEMENT>
```

The screenshot shows a Sublime Text editor window with a C program on the left and its parser output on the right. The C program is as follows:

```
1 {
2     int i
3     i = 0
4
5     int j
6     j = 0
7
8     while (j == 0) {
9         i = 1 + i
10
11         if (i == 3) {
12             j = 1
13         }
14
15         print(1)
16         print(i)
17         print(2)
18         print(j)
19         print(3)
20     }
21
22     print(4)
23 }
24
```

The parser output on the right shows the following steps:

```
< . , RIGHTPAREN>
< print , PRINTSTATEMENT>
< ( , LEFTPAREN>
< 2 , DIGIT>
< . , RIGHTPAREN>
< print , PRINTSTATEMENT>
< ( , LEFTPAREN>
< j , ID>
< . , RIGHTPAREN>
< print , PRINTSTATEMENT>
< ( , LEFTPAREN>
< 3 , DIGIT>
< . , RIGHTPAREN>
< . , BLOCKRIGHTCURLY>
< print , PRINTSTATEMENT>
< ( , LEFTPAREN>
< 4 , DIGIT>
< . , RIGHTPAREN>
< . , BLOCKRIGHTCURLY>
Moving on to parse....
Parsing Block..
Expecting <(>
Got (
Parsing Statement..
Parsing VarDecl..
Expecting <type>
Got int
Expecting <id>
Got i
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Assignment Statement..
Expecting <id>
Got i
Expecting <=>
Got =
Parsing Integer Expression
Expecting <digit>
Got 0
Ending Statement
Ending Statement List
Parsing Statement..
Parsing VarDecl..
Expecting <type>
Got int
Expecting <id>
```

The screenshot shows the same Sublime Text editor window with the same C program. The parser output on the right is more advanced, showing the following steps:

```
Got int
Expecting <id>
Got j
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Assignment Statement..
Expecting <id>
Got j
Expecting <=>
Got =
Parsing Integer Expression
Expecting <digit>
Got 0
Ending Statement
Ending Statement List
Parsing Statement..
Parsing While Statement..
Expecting <while>
Got while
Expecting <(>
Got (
Expecting <id>
Got j
Expecting <booleanOp>
Got ==
Parsing Integer Expression
Expecting <digit>
Got 0
Expecting <=>
Got )
Parsing Block..
Expecting <(>
Got (
Parsing Statement..
Parsing Assignment Statement..
Expecting <id>
Got i
Expecting <=>
Got =
Parsing Integer Expression
Expecting <digit>
Got 1
Expecting <+>
Got +
Expecting <id>
Got i
```

The screenshot shows a Sublime Text editor window with a C program on the left and its parser output on the right. The C program is as follows:

```
1 {
2     int i
3     i = 0
4
5     int j
6     j = 0
7
8     while (j == 0) {
9         i = 1 + i
10
11         if (i == 3) {
12             j = 1
13         }
14
15         print(1)
16         print(i)
17         print(2)
18         print(j)
19         print(3)
20     }
21
22     print(4)
23 }
24
```

The parser output on the right shows the following sequence of events:

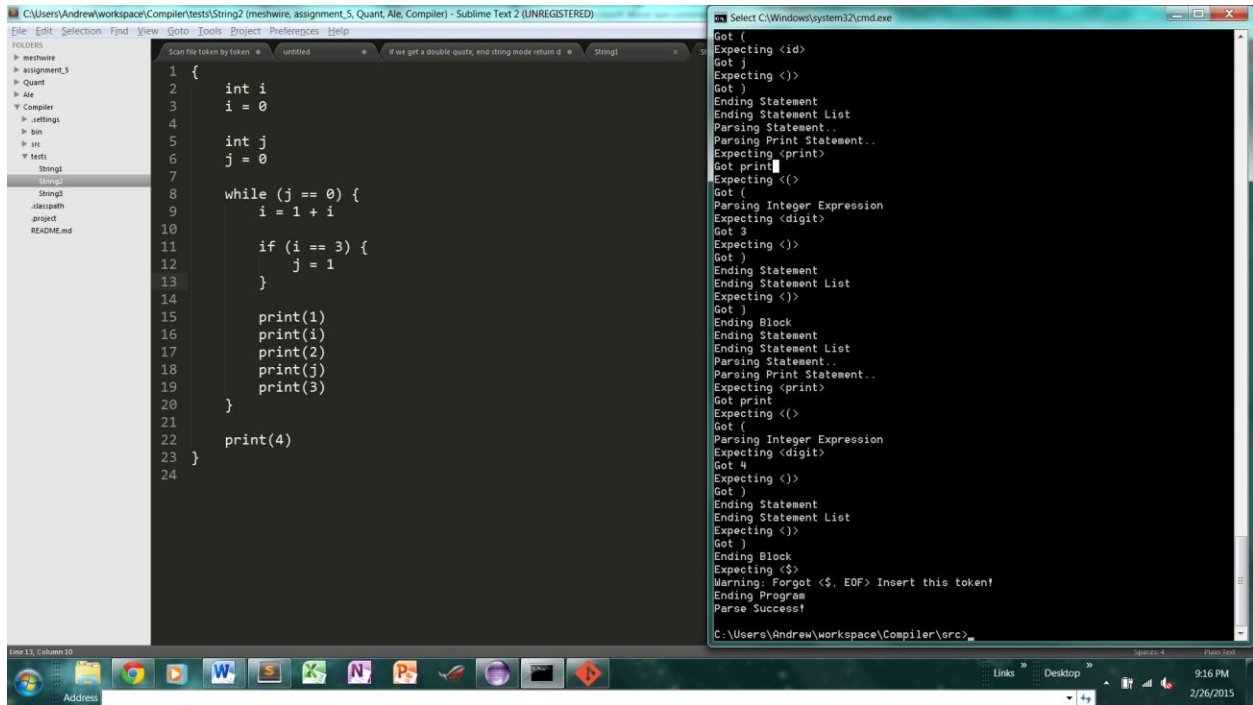
```
Expecting <id>
Got i
Ending Statement
Ending Statement List
Parsing Statement..
Parsing If Statement..
Expecting <if>
Got if
Expecting <(>
Got (
Expecting <id>
Got i
Expecting <booleanOp>
Got ==
Parsing Integer Expression
Expecting <digit>
Got 0
Expecting <)>
Got )
Parsing Block..
Expecting <(>
Got (
Parsing Statement..
Parsing Assignment Statement..
Expecting <id>
Got j
Expecting <=>
Got =
Parsing Integer Expression
Expecting <digit>
Got 1
Ending Statement
Ending Statement List
Expecting <)>
Got )
Ending Block
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Print Statement..
Expecting <print>
Got print
Expecting <(>
Got (
Parsing Integer Expression
Expecting <digit>
Got 1
```

This screenshot is identical to the one above, showing the same C program and parser output. The C program is as follows:

```
1 {
2     int i
3     i = 0
4
5     int j
6     j = 0
7
8     while (j == 0) {
9         i = 1 + i
10
11         if (i == 3) {
12             j = 1
13         }
14
15         print(1)
16         print(i)
17         print(2)
18         print(j)
19         print(3)
20     }
21
22     print(4)
23 }
24
```

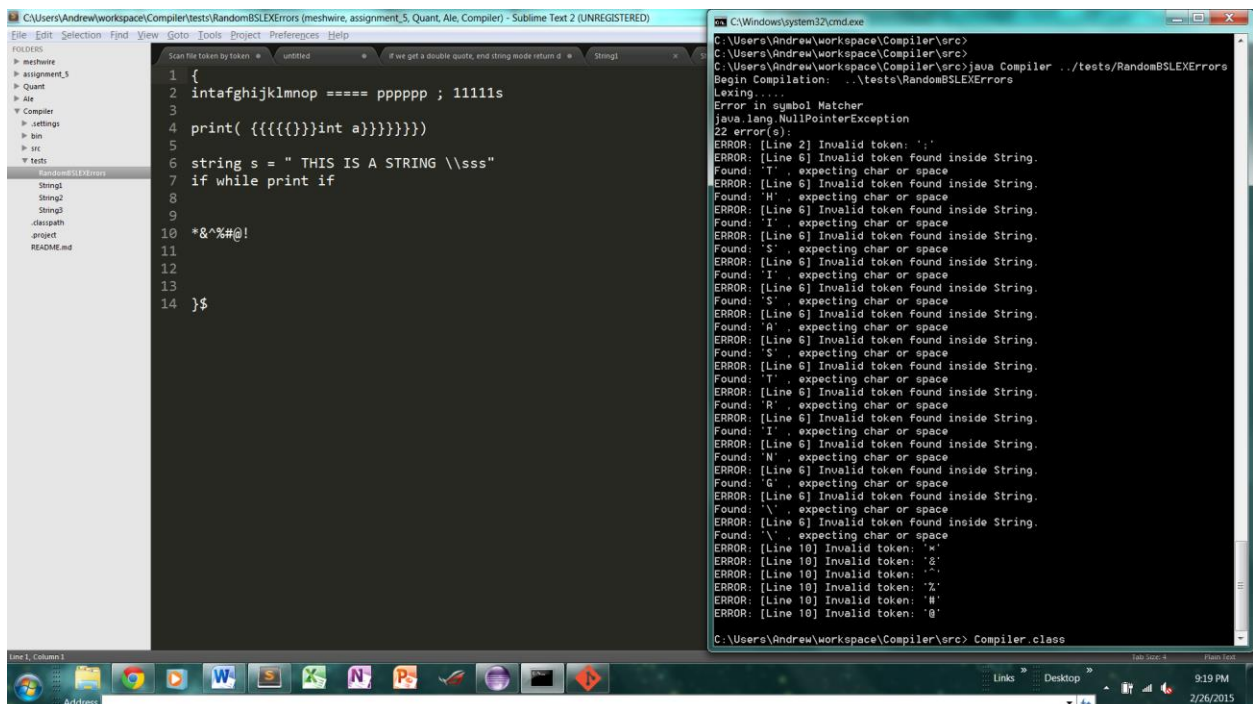
The parser output on the right shows the following sequence of events:

```
Expecting <digit>
Got 1
Expecting <)>
Got )
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Print Statement..
Expecting <print>
Got print
Expecting <(>
Got (
Expecting <id>
Got i
Expecting <)>
Got )
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Print Statement..
Expecting <print>
Got print
Expecting <(>
Got (
Parsing Integer Expression
Expecting <digit>
Got 2
Expecting <)>
Got )
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Print Statement..
Expecting <print>
Got print
Expecting <(>
Got (
Expecting <id>
Got j
Expecting <)>
Got )
Ending Statement
Ending Statement List
Parsing Statement..
Parsing Print Statement..
Expecting <print>
Got print
```



Errors/WhatNot

Lex Errors



Parse Errors

The image is a screenshot of a Windows desktop environment. The primary application is Sublime Text 2, which is open to a Java file named 'ParseError1.java' located at 'C:\Users\Andrew\workspace\Compiler\tests\ParseError1.java'. The code in the editor is as follows:

```
1 {
2
3 int x = 4
4
5
6 }$
```

The left sidebar of the IDE shows a file explorer with the following structure:

- FOLDERS
 - meshwire
 - assignment_5
 - Quant
 - Ale
 - Compiler
 - settings
 - bin
 - src
 - tests
- ParseError1
 - RandomSILEXErrors
 - String1
 - String2
 - String3
 - .classpath
 - .project
 - README.md

The right pane of the IDE displays the output of the Java compiler. It begins with 'Begin Compilation: ..\tests\ParseError1.java Compiler ..\tests\ParseError1', followed by 'Lexing.....' and 'Lex Success!'. The 'TokenStream:' section lists tokens: '<{', 'BLOCKLEFTCURLY>', '<int', 'INTTYPE>', '<x', 'ID>', '<=', 'ASSIGNMENT>', '<4', 'DIGIT>', '<}>', 'BLOCKRIGHTCURLY>', '\$', 'EOF'. The 'Moving on to parse.....' section shows 'Parsing Block..', 'Expecting <{>', 'Got {', 'Parsing Statement..', 'Parsing VarDecl..', 'Expecting <type>', 'Got int', 'Expecting <id>', 'Got x', 'Ending Statement', 'Ending Statement List', 'Expecting <}>', and finally '[Line: 3]ERROR: Unexpected token =', 'Parse Fail.....'. The taskbar at the bottom of the screen shows various application icons (Windows Explorer, Google Chrome, VLC, Word, etc.) and the system clock in the bottom right corner indicates '9:22 PM' on '2/26/2015'.

The image shows a Windows desktop with two application windows. The left window is 'Sublime Text 2 (UNREGISTERED)', which is editing a file named 'ParseError1.mechine'. The code in the editor is:

```
1 {
2
3 int x
4 x =
5
6 }$
```

The file explorer on the left shows a project structure with folders like 'mechine', 'assignment_5', 'Quant', and 'Ale', and files like 'ParserError1.mechine'. The right window is a command prompt titled 'C:\Windows\system32\cmd.exe'. It shows the execution of the Java compiler command: `javac C:\Users\Andrew\workspace\Compiler\src\ParseError1.mechine`. The output shows the compiler successfully processed the file but then threw a `java Compiler ../tests/ParseError1.mechine` command, which resulted in a `ParseException` error: `ParseException: ..\tests\ParseError1.mechine:4: Lexing..... Lex Success! TokenStream: <{ , BLOCKLEFTCURLY> <int , INTTYPE> <x , ID> <x , ID> <= , ASSIGNMENT> < , BLOCKRIGHTCURLY> < , EOF> Moving on to parse..... Parsing Block.. Expecting <{> Got (Parsing Statement.. Parsing VarDecl.. Expecting <type> Got int Expecting <id> Got x Ending Statement Ending Statement List Parsing Statement.. Parsing Assignment Statement.. Expecting <id> Got x Expecting <=> Got = ERROR: Premature end of File Parse Fail....`