Assignment 5: Extended Interpreter

1 Implementation

You are to implement an interpreter for an extended version of the operator language featuring local variables and scoping in Java. The language is again a subset of the Java language. The interpreter is to evaluate the statements in exactly the same way as Java would.

You will implement a Java class `Interpreter` with a main method that reads a file in the given language from standard input (`System.in`) and executes the statements until the program reaches the end of the statement list or fails. The language contains a special `PrintStatement` which should print the value of its expression argument to standard output using `System.out.println`.

If the execution fails, the Interpreter should print the line and column number of the failing operator (`beginLine` and `beginColum` fields of `NodeToken`) with a descriptive error message to standard error. The format of the message should be "LINE:COLUMN MESSAGE".

If the file does not parse, the program should print "Parse error." to standard error (`System.err`).

2 Remarks

Java's static checking will prevent a local variable from being declared twice in any scope. Java also prevents local variables from being potentially accessed without being initialized first. However, since you are writing a simple interpreter, you only need to check that on the execution path no local variable is declared twice or used without being initialized. Hence the following code should not cause an exception for your Interpreter:

```
{
  int i = 42;
  switch(2) {
  case 1:
    int i = 43; // dead!
    break;
```

```
  case 2:
    int j = 44;
  case 3:
    print(j);
  }
}
```

3 Submission

You must submit the implementations to your subversion repository to the directory `3351/$USER/P5/`. Include only the provided grammar, the Interpreter implementation and the provided build script. The files must be called

- `statements.jj`
- `Makefile`
- `src/edu/du/cs/comp3351/p5/Interpreter.java`

You must check that the submitted code compiles by invoking `make`. Verify that the output of your program matches the expected output using your own testcases.