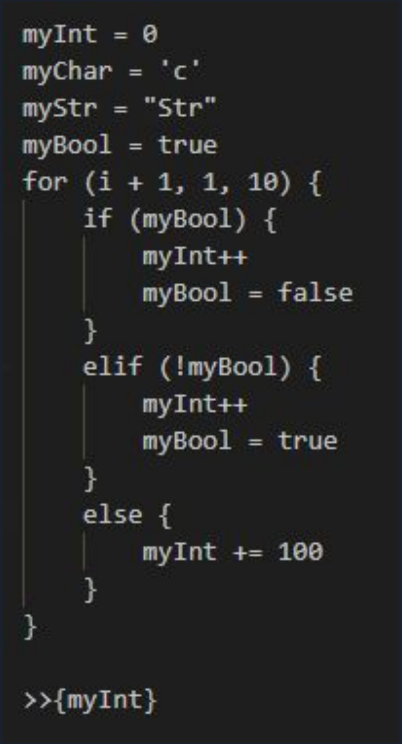
LEMR

Eric Mendoza & Luke Ramirez

Syntax

- Variables are statically typed but use type inference rather than annotations
- Uses Java/C style conditionals, but with Python’s elif instead of else if
- Supports += and ++ operators
- Print is >{} and println is >>{}
- For loops are formatted like
    - for(i * 5, 1, 125) == for(int i = 1; i <= 125; i *= 5) in Java



```
myInt = 0
myChar = 'c'
myStr = "Str"
myBool = true
for (i + 1, 1, 10) {
    if (myBool) {
        myInt++
        myBool = false
    }
    elif (!myBool) {
        myInt++
        myBool = true
    }
    else {
        myInt += 100
    }
}

>>{myInt}
```

Influences on Syntax

- We wanted conditionals to be like Java/C, but disliked having to use the full statement “else if” and opted for elif like Python has
- For loops were somewhat inspired by Python for loops, but we disliked having to use the range() function and “in” keyword for them
- We wanted print statements to be very quick to write out because print statements like System.out.println() not only look cumbersome but also take too long to write out fully
- We wanted statically typed variables because they are nice for checking type errors at compile time, but we dislike having to use annotations for them like in Java

Challenges

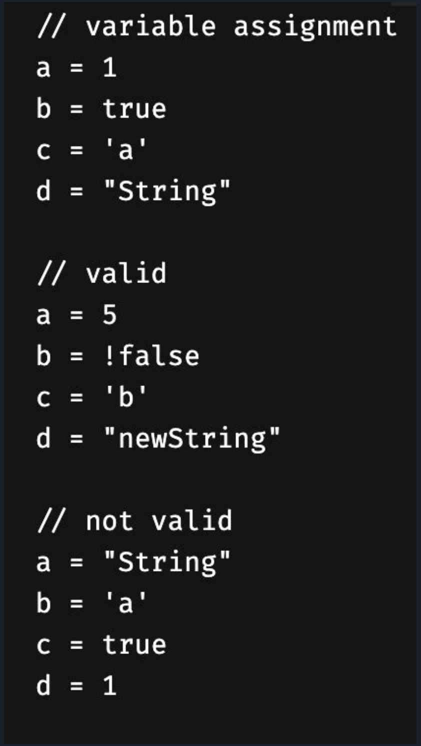
- Getting nested for loops and conditionals to function properly gave us the most trouble because they required complicated parsing to ensure the statement was valid
- Using regex for pattern matching also made it very difficult to make progress because sometimes the regex would seemingly work but then fail on the next thing we tried to match it with
- Boolean expressions and integer expressions were also difficult to properly handle, but less so than conditionals

Tutorial - Index

- Variable Assignment
  - [variable name] = [primitive]
- Conditionals
  - if ([boolean expression]) {[expression]}
  - elif ([boolean expression]) {[expression]}
  - else {[expression]}
- For loop
  - fori ([variable][operator][int], [start index], [end index]) {[expression]}
- Print
  - >{[primitive] or [string literal]or [variable]} - print
  - >>{[primitive] or [string literal] or [variable]} - println

Variables

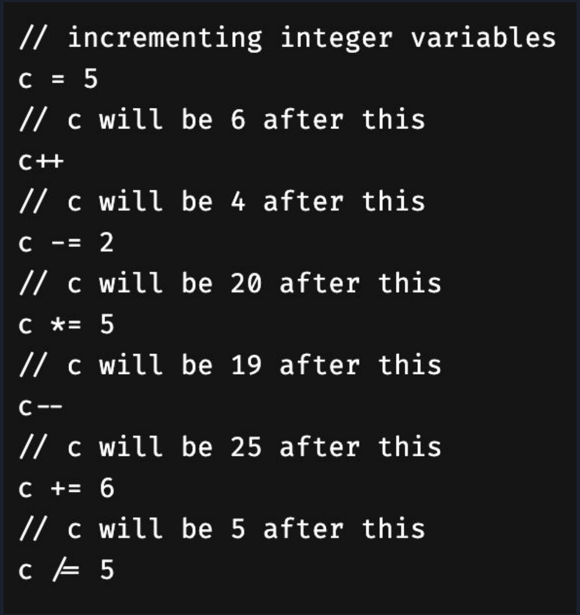
Like other inferred type languages we do not require type annotations, this allows for faster development. However, our language is statically typed so variables cannot change type but are allowed to be reassigned.



```
// variable assignment
a = 1
b = true
c = 'a'
d = "String"

// valid
a = 5
b = !false
c = 'b'
d = "newString"

// not valid
a = "String"
b = 'a'
c = true
d = 1
```

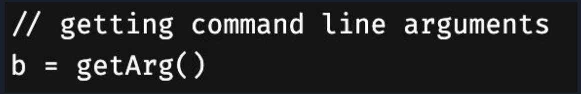


```
// incrementing integer variables
c = 5
// c will be 6 after this
c++
// c will be 4 after this
c -= 2
// c will be 20 after this
c *= 5
// c will be 19 after this
c--
// c will be 25 after this
c += 6
// c will be 5 after this
c /= 5
```

Command line arguments

Our implementation looks like a function call, although we do not have functions in our language.

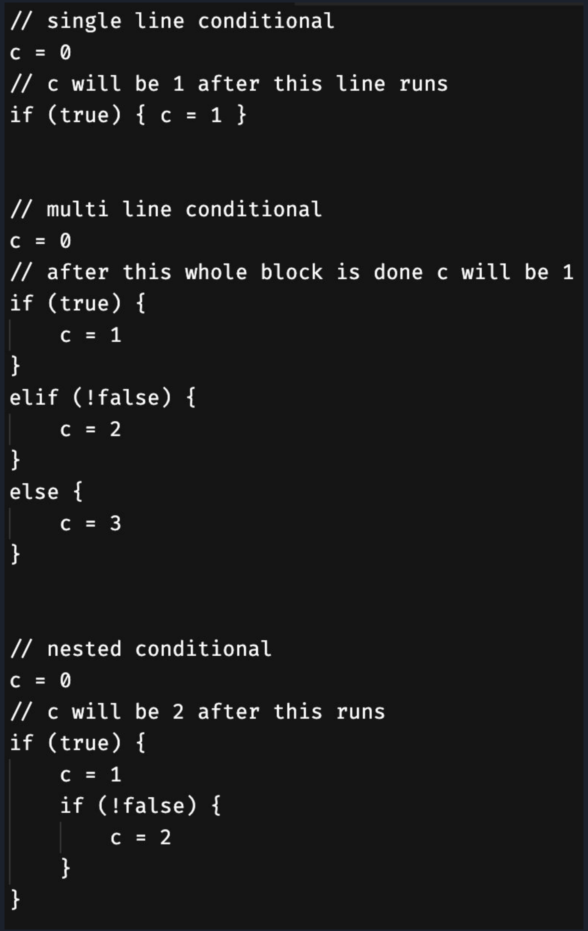
The compiler infers the type of the command line argument and assigns it to the left side like any regular variable assignment



```
// getting command line arguments
b = getArg()
```

Conditionals

Our conditionals borrow from the Java/C style but with Python's elif over the conventional 'else if'. We allow single and multiline expressions as well as nested conditionals.



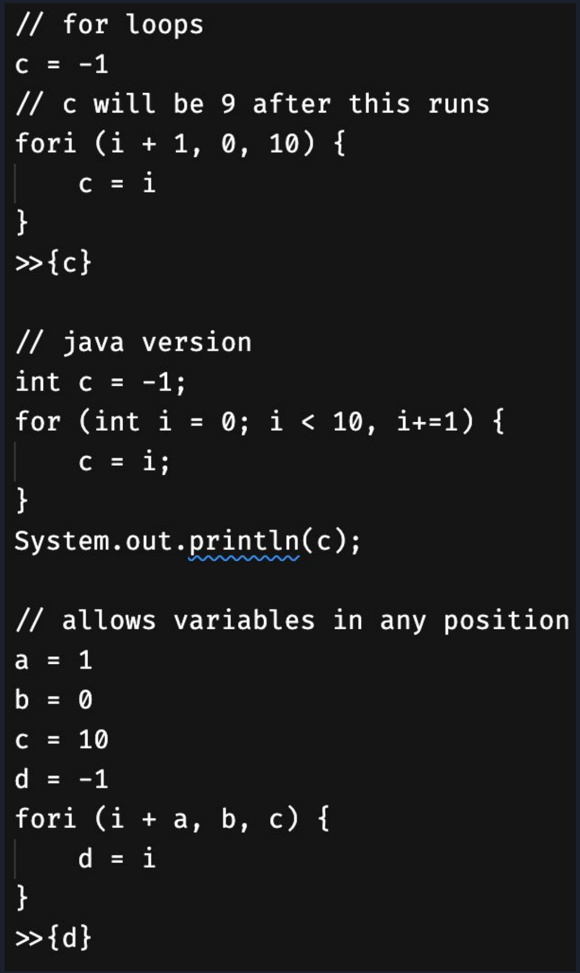
```
// single line conditional
c = 0
// c will be 1 after this line runs
if (true) { c = 1 }


// multi line conditional
c = 0
// after this whole block is done c will be 1
if (true) {
    c = 1
}
elif (!false) {
    c = 2
}
else {
    c = 3
}


// nested conditional
c = 0
// c will be 2 after this runs
if (true) {
    c = 1
    if (!false) {
        c = 2
    }
}
```

Loops

Our for loops are a combination of Java/C for loops and Python. We removed the ‘in’ keyword and range() from Python and kept the curly braces and parentheses from Java.



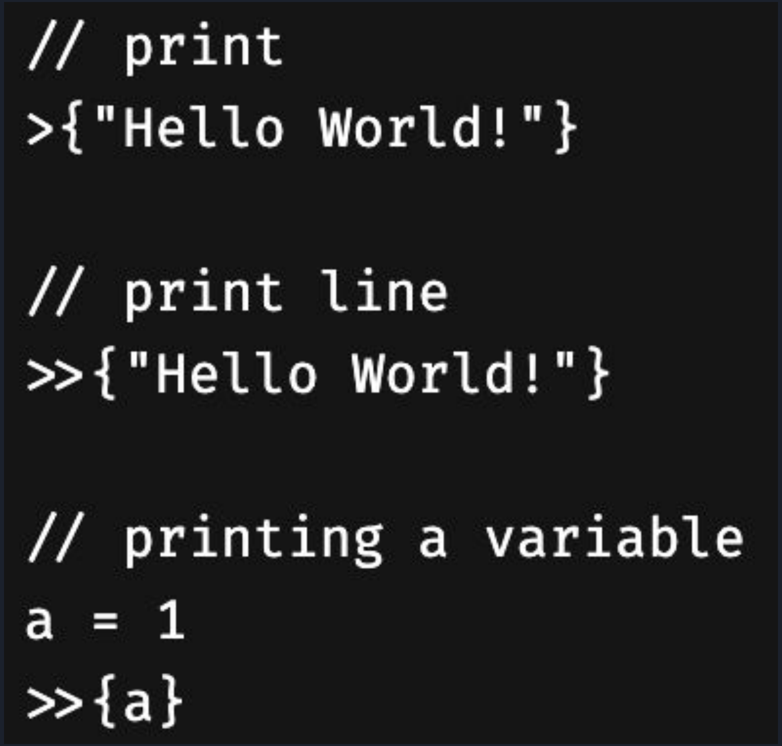
```
// for loops
c = -1
// c will be 9 after this runs
fori (i + 1, 0, 10) {
    c = i
}
>>{c}

// java version
int c = -1;
for (int i = 0; i < 10, i+=1) {
    c = i;
}
System.out.println(c);

// allows variables in any position
a = 1
b = 0
c = 10
d = -1
fori (i + a, b, c) {
    d = i
}
>>{d}
```

Print

We wanted a very simple, fast, and easy way to handle printing because languages like Java which have an unnecessarily long print statement can be clunky to debug in



```
// print
>{"Hello World!"}

// print line
>>{"Hello World!"}

// printing a variable
a = 1
>>{a}
```

How to use and compile our code

Compile all files using “javac *.java” (if you run this command again after creating a [ProgramX.java] file you might get errors compiling)

Then do “java Compiler [ProgramX.txt] [args**]” to print the compiled code to stdout

You could also do “java Compiler [ProgramX.txt] [args**] > [ProgramX.java] && java [ProgramX.java]” instead which allows you to compile and run the code at the same time.

Example:

```
javac *.java

java Compiler Program1.txt 2 5 20
```

Example 2:

```
javac *.java

java Compiler Program2.txt 8 > Program2.java && java Program2.java
```

Exceptions

- We made Java classes for each exception type in our compiler and they all extend a class we made called CompileException
- Exception Types:
  - ArgumentsNotFoundException
  - InvalidBooleanExpressionException
  - InvalidForLoopException
  - InvalidIntegerExpressionException
  - InvalidTypeException
  - InvalidTypeExpressionException
  - MalformedVariableAssignmentException
  - MissingCurlyBraceException
  - TypeMismatchException

Expansion Ideas

- The first thing we would want to add is arrays because they are crucial for data manipulation in any language
- We also thought of adding functions, but we did not have enough time to implement them
- String concatenation and string indexing are things we were hoping to add after arrays
- We named our for loops ‘fori’ because we wanted to implement a ‘foreach’ but that would require at least arrays to be functional