Whats Implemented:
The spark side of the compiler is not implemented.

The parser and interpreter are implemented.

LLVM is partially implemented. Variables, while statements, function calls, closures, add, subtract, less than, less than or equal to, greater than, greater than or equal to, equals, multiply, divide, assignment, and print all work.

Usage:
Running the compiler is simple and only requires a two step method.

1. Type make in the base directory. This will compile the java code.
2. Run the compile shell script to compile a file. This will output an a.out file.
    a. Usage: compile [-emit-llvm] filename

Parser Design:
After using antlr for past projects we both decided we need to find something else to use for the parser. We chose to use Javacc which simplified many things for us. For one it didn't require us to link any outside jars in order to run it. Also, the syntax was much simpler which allowed us to make rapid changes in the structure as we developed it. Inside the javacc grammar file we built the AST as the file was parsed and returned a copy of it upon completion.

Interpreter Design:
Early in the design of the interpreter we decided to take an object oriented approach to its design. For that reason almost everything is considered an expression. We also chose to create a base class called Value that extended expression that represented the different values such as Int or Float. Also, for the base Object class we extended value which gave Object types the ability to handle slots or fields. The interpreter worked by calling a method "Evaluate" upon the root node of the tree returned by the parser. This then calls its children nodes and so on and so forth to evaluate the AST.

LLVM and Static Pass Design:
We continued the object oriented approach to the llvm. We took each llvm instruction that would be generated and created a representation of it in java so we could add the instructions to a list, modify them, and then print them out. We then added functionality into each of the AST nodes from the parser output to add instructions into a list to represent what they do in llvm. We broke the OO design for part of the conversion pass mainly because of some design issue we were faced with when keeping track of things like scope and function calls.

The static pass was written as a treewalker that took the AST as an input. Once the walker was run it contained information pulled from the tree that was relevant for output such as a list of functions and the number of variables in a scope layer. It also added ID's to function and variables for later use. We chose to add scope nodes to the tree during the Parser instead of the static pass since it was easy to add those in at the correct place. The

interpreter functions just basically skip the scope nodes and continue to the children of scope much like it would if the scope wasn't there.

File Statistics:

Filename	Line Count	Description
Expression.java	11	Base expression class that all other interpreter classes extend
Value.java	7	Defines a value expression in the interpreter
Object.java	64	Defines a Object value that other objects can extend
And.java	89	Implements the "&&" expression
IsType.java	52	Checks for the type of the variable
Not.java	62	Implements the "!" expression
OpAdd.java	126	Implements the "+" expression
OpAssign.java	84	Implements the "=" expression
OpDivide.java	139	Implements the "/" expression
OpEquals.java	79	Implements the "==" expression
OpField.java	79	Implements the "." expression
OpFuncDecl.java	39	Implements the function declarations or "funbinds"
OpFunctionCall.java	83	Impelements function calls
OpGreaterThan.java	128	Implements the ">" expression
OpGTE.java	130	Implements the ">=" expression
OpIfElse.java	87	Implements if else blocks
OpInstanceOf.java	42	Implements the instanceof expression for objects
OpLessThan.java	127	Implements the "<" expression
OpLTE.java	130	Implements the "<=" expression
OpMult.java	130	Implements the "*" expression
OpNew.java	85	Implements the "new" expression
OpStringEqual.java	56	Implements the stringequals
OpStringLess.java	69	Implements stringlessthan expression
OpSub.java	129	Implements the "-" expression
OpVarDecl.java	102	Implements variable declaration
OpWhile.java	102	Implements while loops
Or.java	89	Implements the "||" expression
Print.java	71	Implements Print
ReadLine.java	43	Reads in a line
Return.java	31	Implements return
Sequence.java	58	A sequence of instructions
StringLength.java	49	Implements string length
SubString.java	75	Implements a substring check
Environment.java	83	Implements the environment that the interpreter uses for variable storage
ReturnException.java	18	Implements the return exception for use in return statements in the AST

TypeException.java	5	A type exception in the interpreter
Unbound		
Identifier		
Exception.java	4	An unbound identifier in the interpreter
BoolValue.java	36	Implements a Boolean value in the interpreter
ClosureValue.java	61	Implements a closure value in the interpreter
FloatValue.java	42	Implements a float value in the interpreter
Function..java	62	Implements a function in the interpreter
IdValue.java	42	Implements a Id value in the interpreter
IntValue.java	42	Implements a integer value in the interpreter
PlainObject.java	32	Implements a Object value in the interpreter
StringValue.java	50	Implements a String value in the interpreter
VoidValue.java	29	Implements a void value in the interpreter
Scope.java	33	Impelements a scope that is added during the static pass
StaticPass.java	174	The static pass for the program used before LLVM generation
AddInstruction.java	45	LLVM Instruction representations
AssignInstruction.java	32	
BitCastInstruction.java	43	
BoolValueInstruction.java	37	
BranchInstruction.java	62	
CallInstruction.java	52	
Closure.java	53	
CodeGenerator.java	880	
EFrameInstruction.java	29	
EFrame.java	119	
Function		
Declarartion		
Instruction.java	90	
GetElementPtr		
Instruction.java	55	
ICmpInstruction.java	53	
IdValueInstruction.java	33	
IntToPtrInstruction.java	43	
IntValueInstruction.java	28	
LabelInstruction.java	33	
LLVMInstruction.java	45	
LoadInstruction.java	32	
LogicalShiftRight		
Instruction.java	43	
MallocInstruction.java	40	
MultInstruction.java	29	
PhiNodeInstruction.java	34	
PrintInstruction.java	26	

PtrToIntInstruction.java	43	
ReturnInstruction.java	51	
SDivInstruction.java	44	
ShiftLeftInstruction.java	43	
StoreInstruction.java	45	
SubInstruction.java	29	
UDivInstruction.java	44	
VarDeclInstruction.java	23	
VoidInstruction.java	29	
TestAdd.java	66	Test representations of the interpreter nodes
TestIsType.java	92	
TestNot.java	65	
TestOpAdd.java	69	
TestOpAssign.java	54	
TestDivide.java	76	
TestOpEquals.java	52	
TestOpField.java	41	
TestFuncDecl.java	41	
TestFunctionCall.java	41	
TestOpGreaterThan.java	72	
TestOpGTE.java	71	
TestOpIfElse.java	66	
TestOpInstanceOf.java	41	
TestOpLessThan.java	73	
TestOpLTE.java	70	
TestOpMult.java	71	
TestOpNew.java	45	
TestOpStringEqual.java	41	
TestOpStringLess.java	41	
TestOpSub.java	69	
TestOpVarDecl.java	41	
TestOpWhile.java	68	
TestOpOr.java	68	
TestPrint.java	41	
TestReadLine.java	41	
TestReturn.java	41	
TestSequence.java	64	
TestStringLength.java	41	
TestSubString.java	41	
Footle.jj	375	The Javacc grammar file
Total	7725	