Lexical and Syntactic Analyzer Project

This project contains a lexical and syntactic analyzer for a simple language (Kotlin). The project uses Alex for the lexical analyzer and Happy for the syntactic analyzer, processing code examples from a fictional language and generating an abstract syntax tree (AST).

Project Structure

Part one:

The project structure is as follows:

- **Lexer.x:** Lexical analyzer definition using Alex.
- **Parser.y:** Syntactic analyzer definition using Happy.
- **AST.hs:** Definition of the class for the abstract syntax tree (AST).
- **Main.hs:** Main code that runs the parser after executing the lexer the example on exemplos.txt file.
- exemplos.txt : File containing input examples for testing.

Steps to Run:

Generate Lexer.hs:
```
% alex Lexer.x
```
Generate Parser.hs:
```
% happy Parser.y
```
Test the examples:
```
% ghc Main.hs
% ./Main
```

Output example:

[Print (Add (Num 5) (Num 3)),ValDecl "x" (Num 10),Assign "x" (Frase "\"A\""),Print (Num 3),If (EqEq (Id "x") (Num 3)) (Print (Id "x")),IfElse (Lteq (Id "x") (Id "y")) (Print (Id "x")) (Print (Eq (Id "x") (Num 5))),While (Gt (Id "x") (Num 0)) (Print (Num 3)),ValDecl "y" (Add (Num 3) (Mul (Num 2) (Num 4))),ValDecl "z" (And (Bool True) (Bool False))]

Part two:

Project Structure:

- **Lexer.x**: Lexical analyzer definition using Alex.
- **Parser.y**: Syntactic analyzer definition using Happy.
- **AST.hs**: Definition of the class for the abstract syntax tree (AST).
- **TypeChecker.hs**: Implements semantic checks to ensure program correctness.
  - Includes a symbol table that specifies whether variables are mutable or immutable, depending on their declaration (var or val).
  - The symbol table has the following structure: type TypeEnv = Map String (Type, Bool)
    - String: Represents the variable name.
    - Type: Represents the variable's type (e.g., Int, Double, Bool).
    - Bool: Indicates if the variable is mutable (True for var, False for val).
    - Type rules: Variables declared as Int or Double support all standard arithmetic and comparison operations and when operations involve both an Int and a Double, the result is automatically promoted to Double to ensure precision.
- **CodeGen.hs**: Translates the type-checked AST into intermediate code (TAC - Three-Address Code).
- **MIPS.hs**: Converts the intermediate code into MIPS assembly instructions.
- **Main.hs**: Main code that orchestrates the lexer, parser, typechecker, code generator, and outputs MIPS assembly.
- **build**: Bash script to automate the compilation process (using Alex, Happy, and GHC).
- **exemplos.txt**: File containing input examples for testing.

Steps to Run:

```
% chmod +x build
% ./build
```

Output example:

AST:
[ValDecl "x" (Real 10.3),Print (Num 3),If (EqEq (Id "x") (Real 3.5)) (Print (Id "x")),VarDecl "y" (Num 3),IfElse (Lteq (Id "x") (Id "y")) (Print (Id "x")) (Print (Id "y")),While (Gt (Id "x") (Num 0)) (Assign "y" (Add (Id "y") (Real 3.4))),Assign "y" (Add (Real 3.5) (Mul (Num 2) (Num 4))),ValDecl "z" (And (Bool True) (Bool False))]

Symbolic table: (Type, Mutable)
x: (TyDouble,False)
y: (TyDouble,True)
z: (TyBool,False)

GIC:
[MOVEF "t0" 10.3,MOVE "t0" "x",MOVEI "t1" 3,PRINT "t1",MOVE "t2" "x",MOVEF "t3" 3.5,COND "t2" EqEq "t3" "L0" "L1",LABEL "L0",MOVE "t4" "x",PRINT "t4",LABEL "L1",MOVEI "t5" 3,MOVE "t5" "y",MOVE "t6" "x",MOVE "t7" "y",COND "t6" Lteq "t7" "L2" "L3",LABEL "L2",MOVE "t8" "x",PRINT "t8",JUMP "L4",LABEL "L3",MOVE "t9" "y",PRINT "t9",LABEL "L4",LABEL "L5",MOVE "t10" "x",MOVEI "t11" 0,COND "t10" Gt "t11" "L6" "L7",LABEL "L6",MOVE "t13" "y",MOVEF "t14" 3.4,OP Add "t12" "t13" "t14",MOVE "y" "t12",JUMP "L5",LABEL "L7",MOVEF "t16" 3.5,MOVEI "t18" 2,MOVEI "t19" 4,OP Mul "t17" "t18" "t19",OP Add "t15" "t16" "t17",MOVE "y" "t15",MOVE_BOOL "t21" True,MOVE_BOOL "t22" False,OP And "t20" "t21" "t22",MOVE "t20" "z"]

MIPS Code:

```
l.s t0, 10.3
move t0, x
li t1, 3
li $v0, 1
move $a0, t1
syscall
move t2, x
l.s t3, 3.5
bbeq t2, t3, L0
j L1
L0:
move t4, x
li $v0, 1
move $a0, t4
syscall
L1:
li t5, 3
move t5, y
move t6, x
move t7, y
bble t6, t7, L2
j L3
L2:
move t8, x
li $v0, 1
move $a0, t8
syscall
j L4
L3:
move t9, y
li $v0, 1
move $a0, t9
syscall
L4:
L5:
move t10, x
li t11, 0
bbgt t10, t11, L6
j L7
L6:
move t13, y
l.s t14, 3.4
add t12, t13, t14
move y, t12
j L5
L7:
l.s t16, 3.5
li t18, 2
li t19, 4
mul t17, t18, t19
add t15, t16, t17
move y, t15
li t21, 1
```

```
li t22, 0
and t20, t21, t22
move t20, z
```