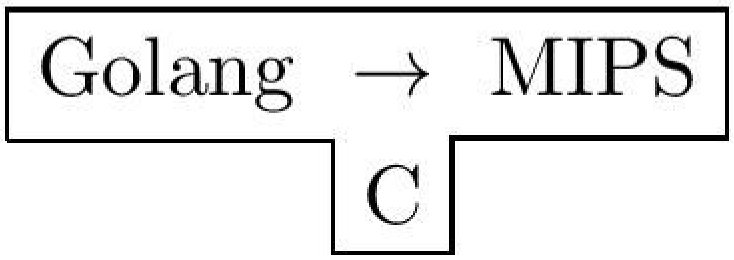
Group Members

1. **Shashank Shekhar (150662) - sshekhar@iitk.ac.in**
2. **Meet Kumar Jigar Sanghvi (150400) - smeet@iitk.ac.in**
3. **Piyush Bansal (150488) - piusbnsl@iitk.ac.in**

==

T Diagram of our Compiler



Source Language - Go (Golang)
Implementation Language - C
Target Language - MIPS

==

BNF for GO

sourceFile ::= packageClause eos (importDecl eos)* (topLevelDecl eos)*
packageClause ::= 'package' IDENTIFIER

importDecl ::= 'import' (importSpec | '(' (importSpec eos)* ')')

importSpec ::= ('.' | IDENTIFIER)? importPath

importPath ::= STRING_LIT

topLevelDecl ::= declaration | functionDecl | methodDecl

declaration ::= constDecl | typeDecl | varDecl constDecl ::= 'const' (constSpec | '(' (constSpec eos)* ')')

constSpec ::= identifierList (type? '=' expressionList)?

identifierList ::= IDENTIFIER (',' IDENTIFIER)*

expressionList ::= expression (',' expression)*

typeDecl ::= 'type' (typeSpec | '(' (typeSpec ';')* ')')

typeSpec ::= IDENTIFIER type

functionDecl ::= 'func' IDENTIFIER (function | signature)

function ::= signature block

methodDecl ::= 'func' receiver IDENTIFIER (function | signature)

Receiver ::= parameters

varDecl ::= 'var' (varSpec | '(' (varSpec eos)* ')')

varSpec ::= identifierList (type ('=' expressionList)? | '=' expressionList)

Block ::= '{' statementList '}'

statementList ::= (statement eos)*

statement ::= declaration | labeledStmt | simpleStmt | goStmt | returnStmt | breakStmt | continueStmt | gotoStmt | fallthroughStmt | block | ifStmt | switchStmt | selectStmt | forStmt | deferStmt

simpleStmt ::= sendStmt | expressionStmt | incDecStmt | assignment | shortVarDecl | emptyStmt

expressionStmt ::= expression

sendStmt ::= expression '<-' expression

incDecStmt ::= expression ('++' | '--')

assignment ::= expressionList assign_op expressionList

```
assign_op ::= ('+' | '-' | '|' | '^' | '*' | '/' | '%' | '<<' | '>>' | '&' | '&^')? '='

shortVarDecl ::= identifierList ':=' expressionList

emptyStmt ::= ';'

labeledStmt ::= IDENTIFIER ':' statement

returnStmt ::= 'return' expressionList?

breakStmt
    : 'break' IDENTIFIER?
    ;

continueStmt
    : 'continue' IDENTIFIER?
    ;

gotoStmt
    : 'goto' IDENTIFIER
    ;

fallthroughStmt
    : 'fallthrough'
    ;

deferStmt
    : 'defer' expression
    ;
ifStmt
    : 'if' (simpleStmt ';')? expression block ( 'else' ( ifStmt | block ) )?
    ;

switchStmt
    : exprSwitchStmt | typeSwitchStmt
    ;

exprSwitchStmt
    : 'switch' ( simpleStmt ';' )? expression? '{' exprCaseClause* '}'
    ;
```

```
exprCaseClause
    : exprSwitchCase ':' statementList
    ;

exprSwitchCase
    : 'case' expressionList | 'default'
    ;

typeSwitchStmt
    : 'switch' ( simpleStmt ';' )? typeSwitchGuard '{' typeCaseClause* '}'
    ;
typeSwitchGuard
    : ( IDENTIFIER ':=' )? primaryExpr '.' '(' 'type' ')'
    ;
typeCaseClause
    : typeSwitchCase ':' statementList
    ;
typeSwitchCase
    : 'case' typeList | 'default'
    ;
typeList
    : type ( ',' type )*
    ;


selectStmt
    : 'select' '{' commClause* '}'
    ;
commClause
    : commCase ':' statementList
    ;
commCase
    : 'case' ( sendStmt | recvStmt ) | 'default'
    ;
recvStmt
    : ( expressionList '=' | identifierList ':=' )? expression
    ;

forStmt
    : 'for' ( expression | forClause | rangeClause )? block
    ;
```

```
forClause
    : simpleStmt? ';' expression? ';' simpleStmt?
    ;

rangeClause
    : (expressionList '=' | identifierList ':=' )? 'range' expression
    ;

//GoStmt = "go" Expression .
goStmt
    : 'go' expression
    ;

type
    : typeName
    | typeLit
    | '(' type ')'
    ;

typeName
    : IDENTIFIER
    | qualifiedIdent
    ;

//TypeLit   = ArrayType | StructType | PointerType | FunctionType | InterfaceType |
//           SliceType | MapType | ChannelType .
typeLit
    : arrayType
    | structType
    | pointerType
    | functionType
    | interfaceType
    | sliceType
    | mapType
    | channelType
    ;


arrayType
    : '[' arrayLength ']' elementType
    ;
```

```
arrayLength
    : expression
    ;

elementType
    : type
    ;

pointerType
    : '*' type
    ;

interfaceType
    : 'interface' '{' ( methodSpec eos )* '}'
    ;

//SliceType = "[" "]" ElementType .
sliceType
    : '[' ']' elementType
    ;

//MapType     = "map" "[" KeyType "]" ElementType .
//KeyType     = Type .
mapType
    : 'map' '[' type ']' elementType
    ;

//ChannelType = ( "chan" | "chan" "<-" | "<-" "chan" ) ElementType .
channelType
    : ( 'chan' | 'chan' '<-' | '<-' 'chan' ) elementType
    ;

methodSpec
    : IDENTIFIER signature
    | typeName
    ;


functionType
    : 'func' signature
    ;
```

```
signature
    : parameters result?
    ;

result
    : parameters
    | type
    ;

parameters
    : '(' ( parameterList ','? )? ')'
    ;

parameterList
    : parameterDecl ( ',' parameterDecl )*
    ;

parameterDecl
    : identifierList? '...'? type
    ;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Operands

//Operand     = Literal | OperandName | MethodExpr | "(" Expression ")" .
//Literal     = BasicLit | CompositeLit | FunctionLit .
//BasicLit    = int_lit | float_lit | imaginary_lit | rune_lit | string_lit .
//OperandName = identifier | QualifiedIdent.

operand
    : literal
    | operandName
    | methodExpr
    | '(' expression ')'
    ;

literal
    : basicLit
    | compositeLit
    | functionLit
    ;
```

```
basicLit
    : INT_LIT
    | FLOAT_LIT
    | IMAGINARY_LIT
    | RUNE_LIT
    | STRING_LIT
    ;

operandName
    : IDENTIFIER
    | qualifiedIdent
    ;

//QualifiedIdent = PackageName "." identifier .
qualifiedIdent
    : IDENTIFIER '.' IDENTIFIER
    ;

//CompositeLit  = LiteralType LiteralValue .
//LiteralType   = StructType | ArrayType | "[" "..." "]" ElementType |
//                SliceType | MapType | TypeName .
//LiteralValue  = "{" [ ElementList [ "," ] ] "}" .
//ElementList   = KeyedElement { "," KeyedElement } .
//KeyedElement  = [ Key ":" ] Element .
//Key           = FieldName | Expression | LiteralValue .
//FieldName     = identifier .
//Element       = Expression | LiteralValue .

compositeLit
    : literalType literalValue
    ;

literalType
    : structType
    | arrayType
    | '[' '...' ']' elementType
    | sliceType
    | mapType
    | typeName
    ;
```

```
literalValue
    : '{' ( elementList ','? )? '}'
    ;

elementList
    : keyedElement (',' keyedElement)*
    ;

keyedElement
    : (key ':')? element
    ;

key
    : IDENTIFIER
    | expression
    | literalValue
    ;

element
    : expression
    | literalValue
    ;

//StructType     = "struct" "{" { FieldDecl ";" } "}" .
//FieldDecl      = (IdentifierList Type | AnonymousField) [ Tag ] .
//AnonymousField = [ "*" ] TypeName .
//Tag            = string_lit .
structType
    : 'struct' '{' ( fieldDecl eos )* '}'
    ;

fieldDecl
    : (identifierList type | anonymousField) STRING_LIT?
    ;

anonymousField
    : '*'? typeName
    ;

//FunctionLit = "func" Function .
functionLit
    : 'func' function
    ;

//PrimaryExpr =
//        Operand |
//        Conversion |
//        PrimaryExpr Selector |
//        PrimaryExpr Index |
//        PrimaryExpr Slice |
//        PrimaryExpr TypeAssertion |
//        PrimaryExpr Arguments .
//
//Selector       = "." identifier .
//Index          = "[" Expression "]" .
//Slice          = "[" ( [ Expression ] ":" [ Expression ] ) |
//                     ( [ Expression ] ":" Expression ":" Expression )
//                 "]" .
//TypeAssertion  = "." "(" Type ")" .
//Arguments      = "(" [ ( ExpressionList | Type [ "," ExpressionList ] ) [ "..." ] [ "," ] ] ")" .

primaryExpr
    : operand
    | conversion
    | primaryExpr selector
    | primaryExpr index
    | primaryExpr slice
    | primaryExpr typeAssertion
        | primaryExpr arguments
    ;

selector
    : '.' IDENTIFIER
    ;

index
    : '[' expression ']'
    ;

slice
    : '[' (( expression? ':' expression? ) | ( expression? ':' expression ':' expression )) ']'
    ;

typeAssertion
    : '.' '(' type ')'
    ;

arguments
    : '(' ( ( expressionList | type ( ',' expressionList )? ) '...'? ','? )? ')'
    ;

//MethodExpr     = ReceiverType "." MethodName .
//ReceiverType  = TypeName | "(" "*" TypeName ")" | "(" ReceiverType ")" .
methodExpr
    : receiverType '.' IDENTIFIER
    ;

receiverType
    : typeName
    | '(' '*' typeName ')'
    | '(' receiverType ')'
    ;

//Expression = UnaryExpr | Expression binary_op Expression .
//UnaryExpr  = PrimaryExpr | unary_op UnaryExpr .

expression
    : unaryExpr
//    | expression BINARY_OP expression
    | expression ('||' | '&&' | '==' | '!=' | '<' | '<=' | '>' | '>=' | '+' | '-' | '|' | '^' | '*' | '/' | '%' | '<<' | '>>' | '&' |
'&^') expression
    ;

unaryExpr
    : primaryExpr
    | ('+'|'-'|'!'|'^'|'*'|'&'|'<-') unaryExpr
    ;

//Conversion = Type "(" Expression [ "," ] ")" .
conversion
    : type '(' expression ','? ')'
    ;

eos
    : ';'
    | EOF
    | {lineTerminatorAhead()}?
    | {_input.LT(1).getText().equals("}") }?
    ;

IDENTIFIER
    : LETTER ( LETTER | UNICODE_DIGIT )*
    ;

KEYWORD
    : 'break'
    | 'default'
    | 'func'
    | 'interface'
    | 'select'
    | 'case'
    | 'defer'
    | 'go'
    | 'map'
    | 'struct'
    | 'chan'
    | 'else'
    | 'goto'
    | 'package'
    | 'switch'
    | 'const'
    | 'fallthrough'
    | 'if'
    | 'range'
    | 'type'
    | 'continue'
    | 'for'
    | 'import'
    | 'return'
    | 'var'
    ;


// Operators

//binary_op  = "||" | "&&" | rel_op | add_op | mul_op .
BINARY_OP
    : '||' | '&&' | REL_OP | ADD_OP | MUL_OP
    ;

//rel_op     = "==" | "!=" | "<" | "<=" | ">" | ">=" .
```

```
fragment REL_OP
    : '=='
    | '!='
    | '<'
    | '<='
    | '>'
    | '>='
    ;

//add_op     = "+" | "-" | "|" | "^" .
fragment ADD_OP
    : '+'
    | '-'
    | '|'
    | '^'
    ;

//mul_op     = "*" | "/" | "%" | "<<" | ">>" | "&" | "&^" .
fragment MUL_OP
    : '*'
    | '/'
    | '%'
    | '<<'
    | '>>'
    | '&'
    | '&^'
    ;

//unary_op   = "+" | "-" | "!" | "^" | "*" | "&" | "<-" .
fragment UNARY_OP
    : '+'
    | '-'
    | '!'
    | '^'
    | '*'
    | '&'
    | '<-'
    ;


// Integer literals
```

```
//int_lit     = decimal_lit | octal_lit | hex_lit .
INT_LIT
    : DECIMAL_LIT
    | OCTAL_LIT
    | HEX_LIT
    ;

//decimal_lit = ( "1" … "9" ) { decimal_digit } .
fragment DECIMAL_LIT
    : [1-9] DECIMAL_DIGIT*
    ;

//octal_lit   = "0" { octal_digit } .
fragment OCTAL_LIT
    : '0' OCTAL_DIGIT*
    ;

//hex_lit     = "0" ( "x" | "X" ) hex_digit { hex_digit } .
fragment HEX_LIT
    : '0' ( 'x' | 'X' ) HEX_DIGIT+
    ;


// Floating-point literals

//float_lit = decimals "." [ decimals ] [ exponent ] |
//            decimals exponent |
//            "." decimals [ exponent ] .
FLOAT_LIT
    : DECIMALS '.' DECIMALS? EXPONENT?
    | DECIMALS EXPONENT
    | '.' DECIMALS EXPONENT?
    ;

//decimals  = decimal_digit { decimal_digit } .
fragment DECIMALS
    : DECIMAL_DIGIT+
    ;

//exponent  = ( "e" | "E" ) [ "+" | "-" ] decimals .
fragment EXPONENT
    : ( 'e' | 'E' ) ( '+' | '-' )? DECIMALS
    ;
```

```
// Imaginary literals
//imaginary_lit = (decimals | float_lit) "i" .
IMAGINARY_LIT
    : (DECIMALS | FLOAT_LIT) 'i'
    ;


// Rune literals

//rune_lit         = "'" ( unicode_value | byte_value ) "'" .
RUNE_LIT
    : '\'' ( UNICODE_VALUE | BYTE_VALUE ) '\''
    ;

//unicode_value    = unicode_char | little_u_value | big_u_value | escaped_char .
fragment UNICODE_VALUE
    : UNICODE_CHAR
    | LITTLE_U_VALUE
    | BIG_U_VALUE
    | ESCAPED_CHAR
    ;

//byte_value       = octal_byte_value | hex_byte_value .
fragment BYTE_VALUE
    : OCTAL_BYTE_VALUE | HEX_BYTE_VALUE
    ;

//octal_byte_value = `\` octal_digit octal_digit octal_digit .
fragment OCTAL_BYTE_VALUE
    : '\\' OCTAL_DIGIT OCTAL_DIGIT OCTAL_DIGIT
    ;

//hex_byte_value   = `\` "x" hex_digit hex_digit .
fragment HEX_BYTE_VALUE
    : '\\' 'x' HEX_DIGIT HEX_DIGIT
    ;

//little_u_value   = `\` "u" hex_digit hex_digit hex_digit hex_digit .
//                           hex_digit hex_digit hex_digit hex_digit .
LITTLE_U_VALUE
    : '\\u' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
    ;
```

```
//big_u_value      = `\` "U" hex_digit hex_digit hex_digit hex_digit
BIG_U_VALUE
    : '\\U' HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT HEX_DIGIT
HEX_DIGIT
    ;

//escaped_char     = `\` ( "a" | "b" | "f" | "n" | "r" | "t" | "v" | `\` | "'" | `"` ) .
fragment ESCAPED_CHAR
    : '\\' ( 'a' | 'b' | 'f' | 'n' | 'r' | 't' | 'v' | '\\' | '\'' | '"' )
    ;


// String literals

//string_lit             = raw_string_lit | interpreted_string_lit .
STRING_LIT
    : RAW_STRING_LIT
    | INTERPRETED_STRING_LIT
    ;

//raw_string_lit         = "`" { unicode_char | newline } "`" .
fragment RAW_STRING_LIT
    : '`' ( UNICODE_CHAR | NEWLINE )* '`'
    ;

//interpreted_string_lit = `"` { unicode_value | byte_value } `"` .
fragment INTERPRETED_STRING_LIT
    : '"' ( UNICODE_VALUE | BYTE_VALUE )* '"'
    ;


//
// Source code representation
//

//letter        = unicode_letter | "_" .
fragment LETTER
    : UNICODE_LETTER
    | '_'
    ;
```

```
//decimal_digit = "0" … "9" .
fragment DECIMAL_DIGIT
    : [0-9]
    ;

//octal_digit   = "0" … "7" .
fragment OCTAL_DIGIT
    : [0-7]
    ;

//hex_digit     = "0" … "9" | "A" … "F" | "a" … "f" .
fragment HEX_DIGIT
    : [0-9a-fA-F]
    ;

//newline = /* the Unicode code point U+000A */ .
fragment NEWLINE
    : [\u000A]
    ;

//unicode_char = /* an arbitrary Unicode code point except newline */ .
fragment UNICODE_CHAR   : ~[\u000A] ;

//unicode_digit = /* a Unicode code point classified as "Number, decimal digit" */ .
fragment UNICODE_DIGIT
 : [\u0030-\u0039]
 | [\u0660-\u0669]
 | [\u06F0-\u06F9]
 | [\u0966-\u096F]
 | [\u09E6-\u09EF]
 | [\u0A66-\u0A6F]
 | [\u0AE6-\u0AEF]
 | [\u0B66-\u0B6F]
 | [\u0BE7-\u0BEF]
 | [\u0C66-\u0C6F]
 | [\u0CE6-\u0CEF]
 | [\u0D66-\u0D6F]
 | [\u0E50-\u0E59]
 | [\u0ED0-\u0ED9]
 | [\u0F20-\u0F29]
 | [\u1040-\u1049]
 | [\u1369-\u1371]
 | [\u17E0-\u17E9]
 | [\u1810-\u1819]
 | [\uFF10-\uFF19]
 ;

//unicode_letter = /* a Unicode code point classified as "Letter" */ .
fragment UNICODE_LETTER
 : [\u0041-\u005A]
 | [\u0061-\u007A]
 | [\u00AA]
 | [\u00B5]
 | [\u00BA]
 | [\u00C0-\u00D6]
 | [\u00D8-\u00F6]
 | [\u00F8-\u021F]
 | [\u0222-\u0233]
 | [\u0250-\u02AD]
 | [\u02B0-\u02B8]
 | [\u02BB-\u02C1]
 | [\u02D0-\u02D1]
 | [\u02E0-\u02E4]
 | [\u02EE]
 | [\u037A]
 | [\u0386]
 | [\u0388-\u038A]
 | [\u038C]
 | [\u038E-\u03A1]
 | [\u03A3-\u03CE]
 | [\u03D0-\u03D7]
 | [\u03DA-\u03F3]
 | [\u0400-\u0481]
 | [\u048C-\u04C4]
 | [\u04C7-\u04C8]
 | [\u04CB-\u04CC]
 | [\u04D0-\u04F5]
 | [\u04F8-\u04F9]
 | [\u0531-\u0556]
 | [\u0559]
 | [\u0561-\u0587]
 | [\u05D0-\u05EA]
 | [\u05F0-\u05F2]
 | [\u0621-\u063A]
 | [\u0640-\u064A]
 | [\u0671-\u06D3]
 | [\u06D5]
 | [\u06E5-\u06E6]
 | [\u06FA-\u06FC]
 | [\u0710]
 | [\u0712-\u072C]
 | [\u0780-\u07A5]
 | [\u0905-\u0939]
 | [\u093D]
 | [\u0950]
 | [\u0958-\u0961]
 | [\u0985-\u098C]
 | [\u098F-\u0990]
 | [\u0993-\u09A8]
 | [\u09AA-\u09B0]
 | [\u09B2]
 | [\u09B6-\u09B9]
 | [\u09DC-\u09DD]
 | [\u09DF-\u09E1]
 | [\u09F0-\u09F1]
 | [\u0A05-\u0A0A]
 | [\u0A0F-\u0A10]
 | [\u0A13-\u0A28]
 | [\u0A2A-\u0A30]
 | [\u0A32-\u0A33]
 | [\u0A35-\u0A36]
 | [\u0A38-\u0A39]
 | [\u0A59-\u0A5C]
 | [\u0A5E]
 | [\u0A72-\u0A74]
 | [\u0A85-\u0A8B]
 | [\u0A8D]
 | [\u0A8F-\u0A91]
 | [\u0A93-\u0AA8]
 | [\u0AAA-\u0AB0]
 | [\u0AB2-\u0AB3]
 | [\u0AB5-\u0AB9]
 | [\u0ABD]
 | [\u0AD0]
 | [\u0AE0]
 | [\u0B05-\u0B0C]
 | [\u0B0F-\u0B10]
 | [\u0B13-\u0B28]
 | [\u0B2A-\u0B30]
 | [\u0B32-\u0B33]
 | [\u0B36-\u0B39]
 | [\u0B3D]
 | [\u0B5C-\u0B5D]
 | [\u0B5F-\u0B61]
 | [\u0B85-\u0B8A]
 | [\u0B8E-\u0B90]
 | [\u0B92-\u0B95]
 | [\u0B99-\u0B9A]
 | [\u0B9C]
 | [\u0B9E-\u0B9F]
 | [\u0BA3-\u0BA4]
 | [\u0BA8-\u0BAA]
 | [\u0BAE-\u0BB5]
 | [\u0BB7-\u0BB9]
 | [\u0C05-\u0C0C]
 | [\u0C0E-\u0C10]
 | [\u0C12-\u0C28]
 | [\u0C2A-\u0C33]
 | [\u0C35-\u0C39]
 | [\u0C60-\u0C61]
 | [\u0C85-\u0C8C]
 | [\u0C8E-\u0C90]
 | [\u0C92-\u0CA8]
 | [\u0CAA-\u0CB3]
 | [\u0CB5-\u0CB9]
 | [\u0CDE]
 | [\u0CE0-\u0CE1]
 | [\u0D05-\u0D0C]
 | [\u0D0E-\u0D10]
 | [\u0D12-\u0D28]
 | [\u0D2A-\u0D39]
 | [\u0D60-\u0D61]
 | [\u0D85-\u0D96]
 | [\u0D9A-\u0DB1]
 | [\u0DB3-\u0DBB]
 | [\u0DBD]
 | [\u0DC0-\u0DC6]
 | [\u0E01-\u0E30]
 | [\u0E32-\u0E33]
 | [\u0E40-\u0E46]
 | [\u0E81-\u0E82]
 | [\u0E84]
 | [\u0E87-\u0E88]
 | [\u0E8A]
 | [\u0E8D]
 | [\u0E94-\u0E97]
 | [\u0E99-\u0E9F]
 | [\u0EA1-\u0EA3]
 | [\u0EA5]
 | [\u0EA7]
 | [\u0EAA-\u0EAB]
 | [\u0EAD-\u0EB0]
 | [\u0EB2-\u0EB3]
 | [\u0EBD-\u0EC4]
 | [\u0EC6]
 | [\u0EDC-\u0EDD]
 | [\u0F00]
 | [\u0F40-\u0F6A]
 | [\u0F88-\u0F8B]
 | [\u1000-\u1021]
 | [\u1023-\u1027]
 | [\u1029-\u102A]
 | [\u1050-\u1055]
 | [\u10A0-\u10C5]
 | [\u10D0-\u10F6]
 | [\u1100-\u1159]
 | [\u115F-\u11A2]
 | [\u11A8-\u11F9]
 | [\u1200-\u1206]
 | [\u1208-\u1246]
 | [\u1248]
 | [\u124A-\u124D]
 | [\u1250-\u1256]
 | [\u1258]
 | [\u125A-\u125D]
 | [\u1260-\u1286]
 | [\u1288]
 | [\u128A-\u128D]
 | [\u1290-\u12AE]
 | [\u12B0]
 | [\u12B2-\u12B5]
 | [\u12B8-\u12BE]
 | [\u12C0]
 | [\u12C2-\u12C5]
 | [\u12C8-\u12CE]
 | [\u12D0-\u12D6]
 | [\u12D8-\u12EE]
 | [\u12F0-\u130E]
 | [\u1310]
 | [\u1312-\u1315]
 | [\u1318-\u131E]
 | [\u1320-\u1346]
 | [\u1348-\u135A]
 | [\u13A0-\u13B0]
 | [\u13B1-\u13F4]
 | [\u1401-\u1676]
 | [\u1681-\u169A]
 | [\u16A0-\u16EA]
 | [\u1780-\u17B3]
 | [\u1820-\u1877]
 | [\u1880-\u18A8]
 | [\u1E00-\u1E9B]
 | [\u1EA0-\u1EE0]
 | [\u1EE1-\u1EF9]
 | [\u1F00-\u1F15]
 | [\u1F18-\u1F1D]
 | [\u1F20-\u1F39]
 | [\u1F3A-\u1F45]
 | [\u1F48-\u1F4D]
 | [\u1F50-\u1F57]
 | [\u1F59]
 | [\u1F5B]
 | [\u1F5D]
 | [\u1F5F-\u1F7D]
 | [\u1F80-\u1FB4]
 | [\u1FB6-\u1FBC]
 | [\u1FBE]
 | [\u1FC2-\u1FC4]
 | [\u1FC6-\u1FCC]
 | [\u1FD0-\u1FD3]
 | [\u1FD6-\u1FDB]
 | [\u1FE0-\u1FEC]
 | [\u1FF2-\u1FF4]
 | [\u1FF6-\u1FFC]
 | [\u207F]
 | [\u2102]
 | [\u2107]
 | [\u210A-\u2113]
 | [\u2115]
 | [\u2119-\u211D]
 | [\u2124]
 | [\u2126]
 | [\u2128]
 | [\u212A-\u212D]
 | [\u212F-\u2131]
 | [\u2133-\u2139]
 | [\u2160-\u2183]
 | [\u3005-\u3007]
 | [\u3021-\u3029]
 | [\u3031-\u3035]
 | [\u3038-\u303A]
 | [\u3041-\u3094]
 | [\u309D-\u309E]
 | [\u30A1-\u30FA]
 | [\u30FC-\u30FE]
 | [\u3105-\u312C]
 | [\u3131-\u318E]
 | [\u31A0-\u31B7]
 | [\u3400]
 | [\u4DB5]
 | [\u4E00]
 | [\u9FA5]
 | [\uA000-\uA48C]
 | [\uAC00]
 | [\uD7A3]
 | [\uF900-\uFA2D]
 | [\uFB00-\uFB06]
 | [\uFB13-\uFB17]
 | [\uFB1D]
 | [\uFB1F-\uFB28]
 | [\uFB2A-\uFB36]
 | [\uFB38-\uFB3C]
 | [\uFB3E]
 | [\uFB40-\uFB41]
 | [\uFB43-\uFB44]
 | [\uFB46-\uFBB1]
 | [\uFBD3-\uFD3D]
 | [\uFD50-\uFD8F]
 | [\uFD92-\uFDC7]
 | [\uFDF0-\uFDFB]
```

```
 | [\uFE70-\uFE72]
 | [\uFE74]
 | [\uFE76-\uFEFC]
 | [\uFF21-\uFF3A]
 | [\uFF41-\uFF5A]
 | [\uFF66-\uFFBE]
 | [\uFFC2-\uFFC7]
 | [\uFFCA-\uFFCF]
 | [\uFFD2-\uFFD7]
 | [\uFFDA-\uFFDC]
 ;

//
// Whitespace and comments
//

WS  :  [ \t]+ -> channel(HIDDEN)
    ;

COMMENT
    :   '/*' .*? '*/' -> channel(HIDDEN)
    ;

TERMINATOR
            : [\r\n]+ -> channel(HIDDEN)
            ;


LINE_COMMENT
    :   '//' ~[\r\n]* -> skip
    ;
```

Please note that the grammar in red will not be included in our language. Moreover, we hope to add the following extra features to our language:

<class declaration> ::= (class modifiers)? “**class”** <IDENTIFIER> <class body>
<class modifiers> ::= <class modifier> | <class modifiers> <class modifier>
<class modifier> ::= **public** | **final**
<try statement> ::= **try** <block> <catches> | **try** <block> <catches>? <finally>
<catches> ::= <catch clause> | <catches> <catch clause>

<catch clause> ::= **catch (** <formal parameter> **)** <block>
<finally > ::= **finally** <block>
<class body declarations> ::= <class body declaration> | <class body declarations> <class body declaration>
<class body declaration> ::= <class member declaration> | <static initializer> | <constructor declaration>
<class member declaration> ::= <field declaration> | <method declaration>
<field declaration> ::= <field modifiers>? <type> <variable declarators> **;**
<field modifiers> ::= <field modifier> | <field modifiers> <field modifier>
<field modifier> ::= **public** | **protected** | **private** | **static**

The new constructs that we have added are :

1. Classes - These are the same as in Java, and will enable encapsulation using private, protected and public members.
2. Try -catch.
    a. The statements within try block are those which have a tendancy to generate an exception during run-time.
    b. The exception is caught in catch block.
    c. The tasks to be done in case an exception occurs are written in the finally block.

The tools that we plan on using in our project are:

1. Lex
2. YACC

We might also use other tools (we do not have much idea right now).