Bhanu Teja P
(201501212)

Compilers Project

Flat-B Language Description:

Flat-B language has two primarily to blocks – Declaration Block and Code Block. Declaration Block has only declarations of variables where as Code Block contains actual code.

Flat-B Syntax:

1. **Data Types**:

Inegers and Array of Integers.

Format:
int <var>;
int <var> [<num>];

examples:
int data, array[100], sum;
int sum;

2. **Declaration Block**:

All the variables have to be declared in the declblock{....} before being used. Multiple variables can be declared in the statement and each declaration statement ends with a semi-colon.

3. **Code Block**:

In the codeblock{...}, there are many statements. Each statement should end with a semi-colon.

4. **Statements:**
There are different types of statements like Assignment Statement, Labeled Statement, If Statement, For Statement, While Statement, GoTo Statement, Print Statement, PrintLn Statement, Read Statement.

5. **Expressions**:

Addition, Subtraction, Multiplication, Division, Mod operations are supported on integers of any arithmetic expressions. Expressions can also be Boolean Expressions which evaluates to either true or false.

Examples:

```
e = 1 + 2;
e = a - 1;
e = 2*b / a;
e = A[i] * B[j];
e = A[i] / A[i+1];
e = a > b
e = A[i] <= A[j]
```

6. **For Statement**:

Types of for-loops supported

Format:

Type-1:

```
for i = <start_expression>, <end_expression>
{
      .....
}
```

Type-2:

```
for i = <start_expression>, <end_expression>, <step_expression>
{
      .....
}
```

In Type-1 , i will be initialized to value of <start_expression> and i will be increased by 1 after each lteration, if i exceeds the value of <end_expression> , iteration willl stop.

Type-2 is same as Type-1 except that after each iteration i will be increaed by the value of <step_expression>.

7. **If-Else Statement**:

Format:

```
if <cond> {
      ....
}

if <cond> {
      ...
}
else {
      ....
}
```

8. **While Statement**:

Format:

```
while <cond> {

}
```

9. **Print Statement**:

Format:

```
print <variable>;
print <string>;
print <string>, <variable>;
print <string>, <variable>, <variable>;
println <string>;
```

10. **Read Statement**:

Format:

```
read <variable>;
```

11. **GoTo Statement:**

Format:

```
goto <label> if <cond>;
goto <label>;
```

12. **Labeled Statement:**

Format:

```
<label> <statements>
```

Flat-B Semantics:

- All the variables must be declared before hand for the usage.
- Only integer data type is supported for all variables and arrays.
- Size of array should be a constant integer.
- In For loop intial value of loop_variable should always be less than or equal to final value and if there is a step value, it should be evaluated to an integer
- Condition for while loop and if should be evaluated to boolean
- GoTo Statement can also be unconditional and if it is conditional then that condition should be evaluated to boolean.

Flat-B CFG:

```
Program             : Decl_Block Code_Block
                    ;

Decl_Block          : DLB OB Decl_List CB
                    ;

Decl_List           :
                    | Decl_List Declaration
                    ;

Declaration         : INT Vars SC
                    ;

Vars                : Var
                    | Vars COMMA Var
                    ;


Var                 : ID
                    | ID OSB NUM CSB
                    | ID OSB Expr CSB
                    ;

Code_Block          : CDB OB Stat_List CB
                    ;

Stat_List           :
                    | Stat_List Statement
                    ;

Statement           : Assignment
                    | LABEL Stat_List  }
                    | IF Boolean_Expr OB Stat_List CB
                    | IF Boolean_Expr OB Stat_List CB ELSE OB Stat_List CB
                    | FOR Lhs EQ Expr COMMA Expr OB Stat_List CB
                    | FOR Lhs EQ Expr COMMA Expr COMMA Expr OB
Stat_List CB
                    | WHILE Boolean_Expr OB Stat_List CB
                    | GOTO ID SC
                    | GOTO ID IF Boolean_Expr SC
                    | PRINT Lhs SC
                    | PRINT STRING SC
                    | PRINT STRING COMMA Lhss SC
                    | PRINTLN STRING SC
                    | READ Lhs SC
                    ;

Assignment          : Lhs EQ Expr SC
                    ;
```

```
Lhs             : ID
                | ID OSB Expr CSB
                ;

Lhss            : Lhs
                | Lhss COMMA Lhs
                ;

Expr            : Lhs
                | BinaryExpr
                | SUB Expr
                | Boolean_Expr
                | NUM
                ;

BinaryExpr      : Expr ADD Expr
                | Expr SUB Expr
                | Expr MUL Expr
                | Expr DIV Expr
                | Expr MOD Expr
                ;

Boolean_Expr    : TRUE
                | FALSE
                | Expr GT Expr
                | Expr LT Expr
                | Expr LEQ Expr
                | Expr GEQ Expr
                | Expr EQUAL Expr
                | Expr NOT_EQUAL Expr
                ;
```

Flat-B CFG examples:

- v = 1 + 2;

  => <ID> EQ <NUM> ADD <NUM> SC

  => <LHS> EQ <Expr> ADD <Expr> SC

  => <LHS> EQ <BinaryExpr> SC

  => <LHS> EQ <Expr>

  => <Assignment>

  => <Statement>

- if a == b {
  print “a b are Equal”;
  }
  else {
  print “a b are Not Equal”;
  }

  => IF <ID> EQUAL <ID> OB
  PRINT <STRING> SC
  CB
  ELSE
  PRINT <STRING> SC
  CB

  => IF <Boolean_Expr> OB <Statement> CB ELSE <Statement> SC

  => IF <Boolean_Expr> OB <Stat_List> CB ELSE <Stat_List> SC

  => <Statement>

- while i < 100 {
  print i;
  }

  => WHILE <ID> LT <NUM> OB
  PRINT <ID> SC
  CB

  => WHILE <Lhs> LT <Expr> OB
  <Statement>
  CB

  => WHILE <Expr> LT <Expr> OB <Stat_List> CB

  => WHILE <Boolean_Expression> OB <Stat_List> CB

  => <Statement>

Flat-B Abstract Syntax Tree Design:

All nodes in abstract syntax are of type <ASTNode>.

Each of the classes get inherits from <ASTNode>.

<ASTNode>
|
|----<Program>
|
|----<Declarations>
|

|----<Declaration>
|
|----<Vars>
|
|----<Var>
|
|----<Statements>
|
|----<Statement>
|
|-----<LHSs>
|
|-----<LHS>
|
|-----<Expr>

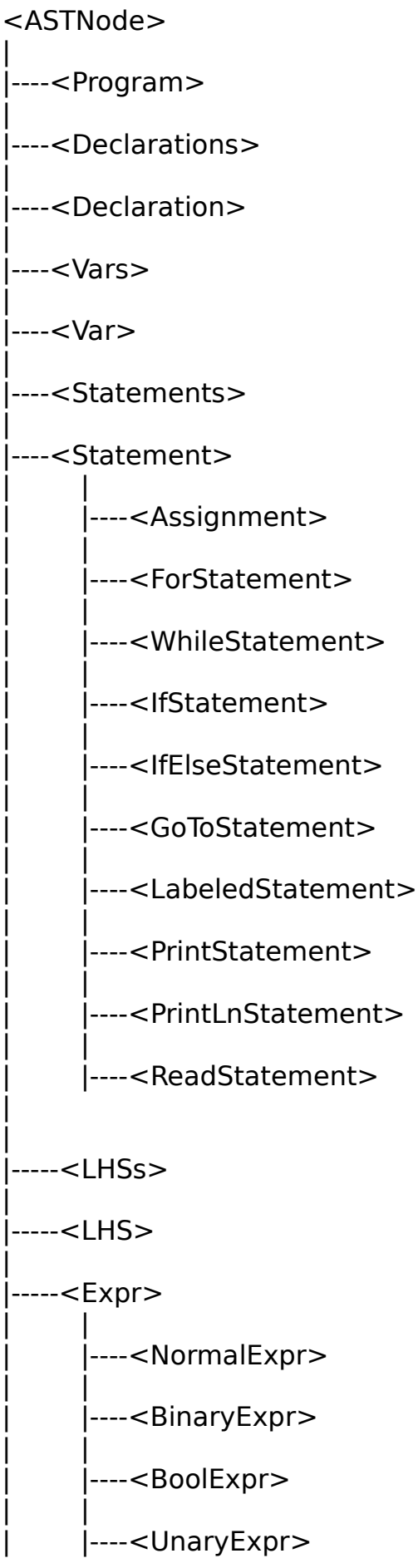
Each of the following classes inherits <Statement>.

<Statement>
|
|----<Assignment>
|
|----<ForStatement>
|
|----<WhileStatement>
|
|----<IfStatement>
|
|----<IfElseStatement>
|
|----<GoToStatement>
|
|----<LabeledStatement>
|
|----<PrintStatement>
|
|----<PrintLnStatement>
|
|----<ReadStatement>

Each of the following classes inherits <Expr>.

<Expr>
|
|----<NormalExpr>
|
|----<BinaryExpr>
|
|----<BoolExpr>
|
|----<UnaryExpr>

Complete AST Class Hierarchy



AST Tree in XML format:

```
<program>
	<declarations>
		<declaration>
			<vars>
				<var />
				<var />
			</vars>
		</declaration>
		<declaration>
			<vars>
				<var />
				<var />
			</vars>
		</declaration>
	</declarations>

	<statements>
		<statement>
			<Assignment>
				<LHS />
				<Expr>
					<Binary Expression />
				</Expr>
			</Assignment>
		</statement>
		<statement>
			<Labeled Statement>
				<Label />
				<Statements></Statements>
			</Assignment>
		</statement>
		<statement>
			<IfStatement>
				<Expr>
					<BooleanExpression />
				</Expr>
				<IfBlock>
					<Statements></Statements>
				</IfBlock>
			</IfStatement>
		</statement>
		<statement>
			<IfElseStatement>
				<Expr>
					<BooleanExpression />
				</Expr>
				<IfBlock>
					<Statements></Statements>
				</IfBlock>
				<ElseBlock>
```

```
                        <Statements></Statements>
                    </ElseBlock>
                </IfElseStatement>
            </statement>
            <statement>
                <ForStatement>
                    <LHS />
                    <CondExpr>
                        <BooleanExpression />
                    </CondExpr>
                    <StartExpr>
                        <BinaryExpression />
                    </StartExpr>
                    <EndExpr>
                        <BinaryExpression />
                    </EndExpr>
                    <StepExpr>
                        <BinaryExpression />
                    </StepExpr>
                    <ForBlock>
                        <Statements></Statements>
                    </ForBlock>
                </ForStatement>
            </statement>
            ........
            ........
            ........
        </statements>
</program>
```

Flat-B Interpreter Design:

Each node in AST has a method interpret

```
int <ClassName>::interpret()
```

Flat-B Interpreter parses the grammar, interprets each node and runs corresponding action in C++.

Interpreter has three symbol tables for variables, array

variable symbol table

```
map <string, int> normal_vars
```

array symbol table

```
map <string, vector <int> > array_vars
```

labeled blocks symbol table

map <string, class Statements* > labeled_blocks

Program::interpret() ----> interprets declarations and statements

Declarations::interpret() ----> interpret each declaration

Declaration::interpret() ----> interprets vars

Vars::interpret() ----> each var present in declaration

Statements::interpret() ----> interprets each statement

Statement::interpret() ----> interprets differently for different type of statement

Expr::interpret() ----> interprets differently for different type of expr

Traversal:
Each node in AST has a method traverse

int <ClassName>::traverse()

This traverse method fills the symbol table and prints the AST Tree in XML format.