}

COMPILERS REPORT

PAAWAN GUPTA
201501046

Flat-B Programming Language Specifications :-

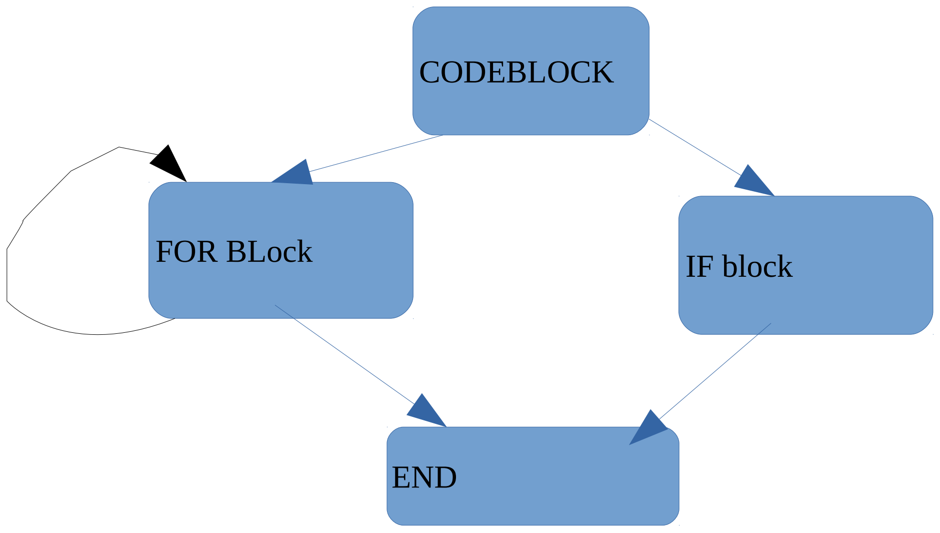
All the variables have to be declared in the declblock{....} before being used in the codeblock{...}. Multiple variables can be declared in the statement and each declaration statement ends with a semi-colon.

```
declblock{
      int x , y , ar[101];
      int m;
}
codeblock {
      x = y+8;
      for i=1,2,10 {
       //Statements
      }
      for i=1,10 {
      ....
      }

      while expression {
      ....
      }

      if expression {
      ....
      }
      ....

label:     if expression {
      ...
      }
      else {
      ....
      }
//    conditional and unconditional goto
      goto label;
      goto label if expression;
      print "blah...blah", val;  //print with a space after
      println "new line at the end";
 // print a line after every string or variable
      read sum;
      read data[i];
}
```

Expressions are evaluated with arithematic expressions first then evaluates AND and then OR .

Syntax and Semantics

As you can see the parser, program consists of declblock and codeblock.
Declblock consists of vector of variables to be declared.
Codeblock consists of vector of statements.
Eg-



Firstly The scanner.l which is a token generator generates the various tokens.
Then parser.y which is a lexical analyser generates an AST by parsing in the rules specified as the grammar of the language.
Thus the AST generated now can be used to generate the IR or can be used for doing interpretation.
Statements can be Assignment/For/While/If-Else/If/Goto/Read/Print
Expressions are of type T1 OR T1
Term are of S1 AND S2
S1 are arithmentic expressions with operator precedence /,*,% > +,-

Design of AST

Ast is designed by creating a programhead through which two child are connected. One is declblock which is a vector of Declarations of variables and the other is codeblock which is a vector of statments. AST consists of private members which contains a number of AST classes objects which then can be called for further traversing. It uses an accept function for traversals.

Visitor Design Pattern and how it is used

First a Interpreter class is designed which consists of function visits.
When we call a child of a node we call it by using the accept function of the child class with the arguments of AST class of the intended AST class.
Accept function is a public method of an Ast class. It then goes to Interpreter class and then searches for the intended visit of that AST class. It then goes to that AST class and performs the actions.

I have used the return functions in some cases for compiling the goto label and expressions.

Design of Interpreter

Using Visitors pattern I have made a Interpreter where I evaluate the Ast functions in the Interpreter function of that AST class. For that i do traversal on the AST tree. After that I go to node and evaluate that. By this I could create an Interpreter.

Design of LLVM Code Generator

It is similar to the Interpreter design. But in this case I have used LLVM IR builder to create an IR which then can be used with lli or llc to run the Flat-B program.

PerFormance Comparsions

CODE	Interpreter	lli	llc
	Wall clock Time	**Wall clock Time**	**Wall clock Time**
Bubble Sort(N=1000)	0.401s	0.017s	0.0033s
Bubble Sort(N=10000)	36.602s	0.289s	0.243s
gcd(N=1000)	0.009s	0.011s	0.000s
gcd(N=10000)	0.025s	0.016s	0.003s
pairsum(N=1000)	0.285s	0.016s	0.003s
pairsum(N=10000)	26.786s	0.133s	0.127s

CODE	Interpreter	lli	llc
	Instructions	**Instructions**	**Instructions**
Bubble Sort(N=1000)	2,72,06,25,306	4,32,55,280	1,31,98,692
Bubble Sort(N=10000)	2,71,98,19,93,018	1,14,99,64,868	1,11,89,57,132
gcd(N=1000)	1,96,74,925	2,85,21,475	12,85,475
gcd(N=10000)	15,27,89,259	3,56,77,392	81,22,419
pairsum(N=1000)	2,01,95,83,682	3,74,88,261	92,35,140
pairsum(N=10000)	2,01,16,86,27,179	83,63,32,257	80,78,73,184