Compiler design

G saisriragh
201501113

<u>Flat B language description:</u>

Flat B programming language is almost similar to C language.This language contains two blocks namely declaration block and code block.In declaration block all the variables to be used in code block have to be declared here.In this language only variables of type integer exist.Normal variables and array variables are possible in this language.All the statements have to be written in the code block.In this programming language functions or classes don’t exist.Some of the statements which are present in the Flat B programming language are assignment,ifelse,while,for ,print,read and goto.

Syntax and semantics:
In this language all the statements including declarations are terminated by a semicolon.
For example:
Int a=1;
a=2+1;

<u>Program syntax</u>

The program constitutes of declaration block and code block and have to be explicitly written in the program.All the statements belonging to that block are represented by using flower brackets.

program: DEC '{' declaration_list '}' COD '{' statement_list '}'

For example :
```
Decl Block{
Int a,b;
}
Codeblock{
/*statements*/
}
```

<u>Decl Block syntax</u>

Declaration block contains of all the declarations of the variables that will be used in the code block.The declaration statement starts with the type.In this language variable of type int are only present.A declaration statement can have declaration of multiple variables as well as arrays.

CFG

declaration_list: decl declaration_list |decl

decl:INT LL SEMI

LL: id ',' LL|id

id:IDENTIFIER
|IDENTIFIER '[' NUMBER ']'

For example:
Int a,b;
Int c[100],d;

<u>Code block</u>

The code block consists of multiple statements such as read,print ,assignment,ifelse,for,while,goto.each of the statements should end with semicolon.Also the variables used in these statements have to be declared in the decl block otherwise it will throw an error

CFG

statement_list: statement_list statement
|statement
Statement:ass| if | for | while |lab | goto |read |print |println

For example
Codeblock
{
a=a+1;
Read b;
Print c;
}

<u>Assignment</u>

For the assignment statement the left side is a variable and right side of the statement can be any expression.The variable can be normal variable or array variable.

CFG

ass:lf '=' expr SEMI

For example
a=(b+5)*c;
data[u]=a;

Read
Read statement reads variables both normal variables and array variable

read:READ lf SEMI

For example:
Read a[u];
Read b;
Read a[c*5+1]

Print
This programming languages has two types of print statements one with a newline at end and other without it.In each of the print statement we can print only variable or string or both.

println:PR1 AD ',' lf SEMI
| PR1 lf SEMI
| PR1 AD SEMI

For example:
Println “sao”,val;
Println val;
Println “sao”;

For loop
For loop syntax is similar to python.

for:FOR IDENTIFIER '=' expr ',' expr bl
|FOR IDENTIFIER '=' expr ',' expr ',' expr bl

The statement list is represented by a block.

For i= expression , expression,expression
{
/*statements*/
}
The first expression is start value.second expression is condition and third expression is increment
Or
For i=expression ,expression
{
/*statements*/

}
The first expression is start value.second expression is condition.The variable will be incremented by 1 default

While loop
While loop is similar to c.It contains an expression and the statlist of while are executed only when the expression is evaluated to true

```
while:WHILE  expr  bl
```

For example

```
while(a>15)
{
/*Statements to be executed*/
}
```

If statement
The are two types of if else statements.one with else and other without else
If the expr evaluates to true then if block will be executed otherwise else block

```
if:IF  expr  bl
| IF  expr  bl ELSE bl
```

```
if(a+b >10)
{

}
Else
{

}
```

Goto
There are two types of goto statements.one is unconditional goto and other is conditional

```
goto: GOTO ab SEMI{$$=new gotos($2);}
```

```
ab:IDENTIFIER IF expr |
IDENTIFIER
```

For example

```
Goto l1 if x>5;
Goto l2;
```

Design of ast

- ❏ Firstly we have class for each of the non terminal symbols in a grammar.

- Class ast is the base class and we have class hierarchy between different classes
- Class ast program which inherits class ast
- Class stat list which inherits class astprogram
- Class stat which inherits class ast
- Class bl which inherits class ast
- Class expr which inherits class ast
- Class bin expr which inherits expr
- Class expr which inherits expr
- Class loc which inherits expr
- Class fors which inherits stat
- Class if which inherits stat
- Class while which inherits stat
- Class read which inherits stat
- Class print which inherits stat
- Class lab which inherits stat
- Class goto which inherits stat
- Class nu which inherits expr

During interpretation we may have to call the child class methods by using ptr to parent class.Hence for this we have virtual functions defined in each of the classes Ast

Visitor design pattern
Visitor design pattern is used to separate the data structure such as ast in this context from algorithms that traverse the code such as interpreter in this case,Using visitor design pattern we can different algorithms for the ast such as interpretation,prefix to postfix conversion,traversal without disturbing the structure of the ast.

Firstly we define a visitor class which contains visit functions for each of the classes of the ast.The visit functions in this base class are abstract .Also we create a accept function in each of the classes in ast which calls the visit class of the visitor which is passed to it as argument.Due to this multiple type of algorithms which involves tree traversals can be done without changing the structure of ast.Basically this visitor class is the base class and we can implement an algorithm of our choice by using creating a class which inherits this visitor class and visit functions are correspondingly defined based on the algorithm being implemented.The visit functions has pointers to classes of ast.

Design of interpreter
Firstly we use visitor design pattern for the interpreter.we create the interpreter class which inherits the visitor class and we define the visit functions which were kept as abstract in the base class.During construction of ast we store the ptrs to the class which are linked to that class using variables of that class.After ast construction we will have ptr to root of the ast tree ie ptr to the ast program class.

- Ast program class has a stalist * st which contains the list of statements in the given flatb program.visit function for ast prog * will have accept function to the coresspondiing statlist
- Statlist class will have a vector of ptrs to each and every statement of the list of statements.visit function for statlist class we will call accept function for each of the ptr to the statements present in the vector.there are 6 types of statements which inherit stat.corresponding accept function will be called
- Assignment class will have the ptr to class of left variable and right expression.Also we have a map which is declared globally which stores the current values of the variables.hence in assignment class visit function we will evaluate the right expression and update the value of left variable with the help of map
- If else class will have the expression,if block which is run if the expr is evaluated and else block visit function for if else class will evaluate the expression and checks whether condition is satisfied or not.If it is satisfied accept function of if block is called otherwise accept function of else block if present will be evaluated
- For class class will have the start expression finnish expression and condition expression.expression will be evaluated and for loop will be executed and each time we update the value of running variable by storing it in map
- While class will have expr and while block.In visit function for this class we will evaluate the expression by calling its accept function and if the condition is satisfied we will call accept function of the while block until the condition fails,
- Visit function for read class will read the input by using cin and updates the value using the map
- Visit function for print class will print the string and value of variable by retrieving it from map
- Goto class has a label name and expr.goto statements is two types.one conditional and other unconditional.Also we have global vector which has ptrs to all the label class.the visit function will call accept function of the label * for which label name matches and based on the expr
- Expr class is base class and will have binexpr ,unexpr,variable and number.Corresponding visit functions will have accept functions for the subsequents and returns the value of that expr after performing all the operations

Design of llvm interpreter
Basically we will have codegen functions to each of the classes for generating llvm.
Firstly we will declare all the variables present in the decl block as global variables in llvm

- Now we will create a main function of return type void and does not take any arguments as input for the function.Also we will create a basic block in which instructions will be stored.
- Llvm interpreter traverses through the ast and generated llvm ir code
- Codegen for statlist class will have codegen to individual statements
- Codegen for assignment class will call codegen for the right expression and stores in the variable using createstore function in llvm

- Codegen for readfunction get the value * of that variable using get Named global and reads the input using The Module->getOrInsertFunction("scanf", FType) and Builder.CreateCall(printfunc,refargs);
- Codegen for print expression loads the value of the variable.Also it prints the string present in print statement using llvm builtin functions
- Codegen for forloop will initialize the start value in the variable and create a new block for the loop and after loop.conditional break instruction will be inserted whether to goto loop block or after loop block depending on the condition.Similarly within the block loop we insert conditional break instruction as mentioned above.
- Codegen for while loop will generate codegen for the while expression.Also we create two blocks loop and after loop insert conditional break instruction based on whlieexpression.In the loop block we generate codegen for the statements within the while loop and similarly create cond break instruction
- Codegen for ifelse stat will generate codegen for the expression and create three blocks :-if block ,else block and contbb and we will insert a condbr instruction to go to if or else block based on the expression.ALso we will insert unconditional br in if block and else block to go to cont block
- Codegen for goto will generate codegen for expr if present and create cond br to the label block if the statement is a conditional goto and create br instruction if it is unconditional goto
- Codegen for expr will generate codegen will generate codegen for subsequent instructions and does arithmetic operations using llvm inbuilt functions

Performance comparison of my interpreter ,lli and llc on the following programs

1)Bubble sort on 5 numbers
Interpreter took 13.6 ms and had 93,62,003 instructions
lli took 19.6 ms and had 2,84,89,471 instructions
Llc took 2 ms and had 6,80,290 instructions

2)finding prime factors for a 2 digit number
Interpreter took 13.67 ms and had 77,97,575 instructions
lli took 14.93 ms and had 2,49,60,475 instructions
Llc took 0.79 ms and had 5,80,649 instructions

3)cumulative sum of an array of size 6
Interpreter took 12.92 ms and had 86,02,597 instructions
lli took 12.6 ms and had 2,54,48,403 instructions
Llc took 1.85 ms and had 6,44,631 instructions

We can observe that for any code llc takes the minimum amount of time to execute the instruction and has minimum no of instructions while execution