Chaitanya Patel 201501071

FlatB Description

- Data Types : Inegers and Array of Integers.

  `int data, array[100];` `int sum;`

- Identifiers contains only alpha-numerical characters starting with an alphabet.
- 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. Variables can't be assigned in declblock.
- Expressions
  - C like expressions
  - Expressions can have arithmetic operators and conditional operators : Both are treated same.
  - Conditional operation returns 1 for true condition and 0 for false condition (flatB supports only integers).
  - Some Examples :
    - `a + b`
    - `a + (b + c) / (x + y) % p`
    - `(a == 0) + (b <; (a + d))`
- for loop
  - `for i = a, b {}` is same as `for(i = a, i <; b, i++)` in C.
  - `for i = a, x, b {}` is same as `for(i = a, i <; b, i+=x)` in C.
- if-else statement

  `if expression {}`

  `if expression {} else {}`
- while statment

  `while expression {}`
- conditional and unconditional goto

  `goto label`

  `goto label if expression`
- print/read

  `print ";blah...blah";, val` // New line at the end of each print `read sum` `read data[i]`

Syntax and Semantics

Keywords

- `int, codeblock, declblock, print, read, if, else, while, for, goto`

Context Free Grammar

program: decl_block code_block

decl_block: DECLBLOCK '{' '}' | DECLBLOCK '{' decl_statement_list '}'

decl_statement_list: decl_statement_list decl_statement | decl_statement

decl_statement: INT decl_variable_list ';' | ';'

decl_variable_list: decl_variable_list ',' IDENTIFIER | decl_variable_list ',' IDENTIFIER '[' INT_LITERAL ']'
 | IDENTIFIER | IDENTIFIER '[' INT_LITERAL ']'

code_block: CODEBLOCK '{' '}' | CODEBLOCK '{' statement_list '}'

statement_block: '{' '}' | '{' statement_list '}'

statement_list: statement_list statement | statement

statement: expression ';'
 | variable '=' expression ';'
 | statement_block
 | IF expression statement_block
 | IF expression statement_block ELSE statement_block
 | FOR variable '=' expression ',' expression statement_block
 | FOR variable '=' expression ',' expression ',' expression statement_block
 | WHILE expression statement_block
 | GOTO IDENTIFIER ';'
 | GOTO IDENTIFIER IF expression ';'
 | READ read_variable_list ';'
 | PRINT printable_list ';'
 | IDENTIFIER ':'
 | ';'

expression: expression '+' expression
 | expression '-' expression
 | expression '*' expression
 | expression '/' expression
 | expression '%' expression
 | expression LESS expression
 | expression LESS_OR_EQUAL expression

| expression GREATER expression

| expression GREATER_OR_EQUAL expression

| expression EQUAL expression

| expression NOT_EQUAL expression

| expression OR expression

| expression AND expression

| '-' expression %prec UMINUS

| '(' expression ')'

| variable

| INT_LITERAL

variable: IDENTIFIER

| IDENTIFIER '[' expression ']'

read_variable_list: read_variable_list ',' variable

| variable

printable_list: printable_list ',' STRING_LITERAL

| printable_list ',' expression

| STRING_LITERAL

| expression

AST Design

Class Hierarchy

- AST_node
    - AST_program
    - AST_decl_block
    - AST_code_block
    - AST_statement
        - AST_expression_statement
            - AST_binary_operator_expression
            - AST_unary_operator_expression
            - AST_variable
                - AST_variable_single_int
                - AST_variable_array_int
            - AST_int_literal
        - AST_assignment_statement

        - AST_block_statement
        - AST_if_statement
        - AST_ifelse_statement
        - AST_for_statement
        - AST_while_statement
        - AST_goto_statement
        - AST_read_statement
        - AST_print_statement
        - AST_label_statement
    - AST_string_literal

Visitor Design Pattern

- Each AST node has an `accept` method which accepts an instance of `Visitor` class.
- `Visitor` class has `visit` method for each AST node.

Traverse

- `Traverse` is derived class from `Visitor` class.
- It traverses on the AST and print its structure in raw format.

Interpreter

Evaluate

- `Evaluate` is derived class from `Visitor` class.
- It traverses on the AST and evaluates the input program.
- It stores normal variables and array variables which are declared in `declblock` as its private members.

Code generator

CodeGen

- `CodeGen` is derived class from `Visitor` class.
- It traverses on the AST and generates LLVM bytecode.
- `declblock` variables are declared as global variables in generated code.
- Code written in `codeblock` is implemented in a `main` function in generated code.
- Since return type of `Visitor` is `int` , it stores `llvm::Value *` in a private variable to return.
- For goto statements, a map from goto labels to Basicblock's starting point is stored.
- Since the language supports only one data type - int and both - arithmetic and conditional expressions , typecasting is done from boolean to int.

Performance

Binary Search

- Array of 5,000,000 numbers sorted
- 5,000,000 Queries

lli interpreter

real 0m1.201s user 0m1.172s sys 0m0.024s

llc static compiler

real 0m1.181s user 0m1.152s sys 0m0.028s

My interpreter

real 1m16.621s user 1m16.124s sys 0m0.472s

Sieve of Eranthoses

- Finding primes up to 5,000,000

lli interpreter

real 0m0.286s user 0m0.264s sys 0m0.020s

llc static compiler

real 0m0.279s user 0m0.268s sys 0m0.008s

My interpreter

real 0m7.556s user 0m7.488s sys 0m0.056s

Bubblesort

- bubblesort over 20,000 numbers

lli interpreter

real 0m0.996s user 0m0.992s sys 0m0.000s

llc static compiler

real 0m0.851s user 0m0.848s sys 0m0.000s

My Interpreter

real 2m10.433s user 2m10.364s sys 0m0.020s