Compiler Design Report

JAVA --

M. Ahmed Hussain (201451009)

Naresh Suman (201451010)

Aditya Raj (201451032)

Tarachand Gurjar (201451032)

Indian Institute of Information Technology,

Vadodara

Course Code : - CS306

Course Name : - Compiler Design

Date : - 21 April 2017

Introduction

Designing a compiler requires time consuming process. Many phases such as designing grammar,creating parser,semantic,generating intermediate code etc.However,used grammar should be unambiguous, context free, and to accept

the operator precedence.This project describes basic about how to built a compiler from scratch and it’s phases ,complexities.

Unambiguity, context free and acceptance of the operator precedence is the major properties of this compiler.many keywords are inherited from java languages to increase the understanding of the user and readability of the code.

Prerequisite Knowledge -

Lex

It is computer program that generate lexical analyzers. It reads an input stream specifying the lexical analyzer and outputs source code implementing the lexer in the C programming language.

Structure of Lex File

The definition section defines tokens and match string according to some regular expression.

Yacc

Yet another compiler compiler(Yacc) is a Look Ahead Left-to-Right(LALR) parser generator which tries to make syntactic sense of the source code based on an analytical grammar. The input to the Yacc is grammar with snippet of java -- code attached to the rules. Structure of Yacc File

The definition section defines macros and import header files written in C. It is also possible to write any C code here, which will be copied into generated source file

The grammar section associates production rules with java -- statements . When the parser sees text in the input matching a given pattern, it will execute associate code.

The C Code section contains C statements and functions that are copied to the generated source file

Key Features

These are the main features of this compiler as follows :--

Arithmetic operations

This compiler supports simple arithmetic operations such as addition, multiplication, substraction, division

Examples :-

```
a=b+c;

a=b-c;

a=b*c;

a=b/c;
```

Conditional Statements

It includes conditional if and else statements.Nested if - else is also implemented in this compiler.

Examples :-

```
if (a==b) {
        printf(“a equal b”);
} else {
printf(“a not equal b”);
}

if (a<=b)
printf(“a less than or equal b”);
```

Control Structures

It supports while loop structures.Nested looping control structures are also included.

Examples :-

```
1.  int  a = 5;
        while (a > 0) {
        b  =  b - 5 ;
        a  = a - 1 ;
     }
```

Return Value By Function

This compiler supports method feature. Function can have single , multiple or null formal parameter , according to this function can return single value.

Ambiguity

Our grammar may be ambiguous because there is may be more than one parse tree for a given expression. Used grammar to build this compiler may have ambiguous property.

Description of Tokens	Meaning associated to it
Main	Start position of program
Extends	It is used for inheritance
New	for new object
Int,double,etc	data types
Class	for generating class
True,false	conditions
while,if,else	Conditional loops
ID	for identifier
Static	static memory allocation

Publc allowance of the data

String for sentances

Grammar

Program : class_dec | class_dec Program

;

class_dec: CLASS SPACE ID '{' var_dec method_dec '}'

| CLASS SPACE ID SPACE EXTENDS SPACE ID '{' var_dec method_dec '}'

;

var_dec : Type SPACE ID ';'

| Type SPACE ID '=' Expr ';'

| var_dec var_dec

|

;

method_dec : PUBLIC SPACE Type SPACE ID '(' ')' '{' var_dec Statement '}'

| PUBLIC SPACE Type SPACE ID '(' FormalParams')' '{' var_dec Statement '}'

| PUBLIC SPACE STATIC SPACE VOID SPACE MAIN '(' STRING '[' ']' SPACE ID ')' '{' var_dec Statement '}'

;

FormalParams : Formal

| FormalParams ',' Formal

;

Formal : Type SPACE ID ;

Type : BasicType

```
    |BasicType'[' ']'
    | ID
    | VOID
    ;
BasicType : BOOLEAN
          | INT
          | DOUBLE
          ;

Statement  : '{' Statement '}'
          | ID '=' Expr ';'
                | ID '[' Expr ']'  '=' Expr ';'
          |Expr '.' ID '=' Expr ';'
          |Expr '.' ID  '[' Expr ']'  '=' Expr ';'

          | ID '('  ')' ';'
                |ID '(' Exprlist ')' ';'
                | Expr '.' ID '('  ')' ';'
                | Expr '.'  ID '(' Exprlist ')' ';'

          | IF '(' Expr ')' Statement ELSE Statement
          |IF '(' Expr ')' Statement
          |WHILE '(' Expr ')' Statement
          |SYSTEMOUT '(' ')' ';'
           |SYSTEMOUT '('SENTENCE')' ';'
```

```
        |SYSTEMOUT '(' Expr ')' ';'
        |RETURN  ';'
         |RETURN Expr ';'  {$$ = $2;}
        ;
Expr :  Expr Binop Expr
        | '!' Expr
        | Expr '[' Expr']'
        | Expr '.' LENGTH  '('  ')'
        |  ID '('  ')'
         |  ID '('  Exprlist  ')'    {;}
         |  Expr '.'  ID '('  ')'    {;}
         |  Expr '.'  ID '('  Exprlist  ')'    {;}
        |  ID     {;}
         |  Expr '.' ID     {;}
        | NEW  BasicType '[' Expr  ']'   {;}
        | NEW ID  '('  ')'        {;}
        | '(' Expr ')'     {;}
        | THIS           {;}
        | Number      {;}
        ;

Exprlist : Expr     {;}
     | Exprlist ',' Expr  {;}
     ;
```

```
Number   : NUM
         | TRUE
         | FALSE
         ;

Binop : '+'
         | '-'
         | '*'
         | '/'
         | AND
         | OR
         | EQ
         | NE
         | '<'
         | LE
         | '>'
         | GE
         ;
```

Grammar Parsing

Our compiler parses key features that we mentioned in grammar production rules. All key features Arithmetic Operation, Conditional Statements, Return value by function. We have somewhat successfully parsed the grammar. But it has still some fail for much complex programs

Grammar Semantics

our compiler supports some basic semantic rules. We designed semantic rules for arithmetic operations and print.It is still in process

Sample Code

```
class adi{

public static void main(String[] a) {

int v;

v=8;
```

```
system.out.println(v);

}

}
```

Commands :

Lex : lex filename.l

Yacc : yacc filename.y

cc -c lex.yy.c y.tab.c

cc -o a2.out lex.yy.o y.tab.o -lfl

./a2.out

Sample Output:

8

Accepted

Conclusion

This compiler is just a basic approach to implement complete compiler. It has many basic functionality that has in any compiler. It also has some limitations at this stage in parsing and semantics ,newline and spaces .Besides this it is good basic approach to learn how to build compiler any language.