LANGUAGE PROCESSOR LAB MAJOR ASSIGNMENT

INTERMEDIATE CODE GENERATION FOR SUBSET C LANGUAGE

Submitted by :

KARNIKA AGARWAL

B.TECH 3rd yr

ROLL NO : 127131 SECTION: A

NATIONAL INSTITUTE OF TECHNOLOGY,WARANGAL

OBJECTIVES :-

The main objective of this assignment is to generate intermediate code for a subset C-like language. Intermediate code must be easy to produce and easy to translate to machine code.I generated intermediate code with the help of Lex and Yacc .Its easy to write code in Lex and Yacc for parsing the input file and checking syntax rules and grammar rules.

My code generates intermediate code for following:-

- Datatypes :integers,float,double,Boolean
- Shorthand Operators: +=,-=,*=,/=
- Unary minus
- Binary Operators: +,-,*,/,@(exponentiation operator)
- Bitwise Operator: &,|,^(XOR)
- Logical Operator: &&,||
- Relational Operator: ==,>=,<=,>,<
- Assignment Statement: =
- Control structures
    -Conditional: if,if-else,switch
    -Repetative: while

Under Guidance of:

Proff. T.RAMESH

NITW

DESCRIPTION:

Lex stands for “Lexical Analyser” ,its main job is to break up an input stream into more usable elements as tokens. YACC stands for "Yet Another Compiler Compiler”. Officially called as Parser.Its job is to analyse the structure of input stream and operate,verifies whether the input stream is syntactically correct.

How exactly my code works is-

Firstly my input file is read by lex which will break the input stream into tokens and pass those tokens to yacc where yacc will parse ,apply grammar rules,if no rule matches then it generates an error ,it evaluate the expression according to associativity and precedence order,for each rule matches,some function is called which will push the current text or token into stack which helps in generating intermediate code.

Steps To Compile And Run:

```
$lex k.l

$yacc k.y

$gcc y.tab.c –ll –ly

$./a.out <input_file>
```

Lex code (k.l) :

```
letter [a-zA-Z]
digit[0-9]
%%
[ \t]                  ;
[ \n]      {yylineno=yylineno+1;}
int        return INT;
main       return MAIN;
float      return FLOAT;
char       return CHAR;
double     return DOUBLE;
void       return VOID;
long       return LONG;
const      return CONST;
typedef    return TYPEDEF;
while      return WHILE;
for        return FOR;
do         return DO;
if         return IF;
else       return ELSE;
break      return BREAK;
continue   return BREAK;
switch     return SWITCH;
case       return CASE;
default    return DEFAULT;
struct     return STRUCT;
return     return RETURN;
printf("(".*")")      ;
{digit}+   return NUM;
```

```
{letter}({letter}|{digit})*  return ID;
"\\n"|"\\b"|"\\t"|"\\a" ;
">="       return GE;
"<="        return LE;
"=="       return EE;
"!="       return NE;
"+="       return AE;
"-="       return SE;
"*="       return ME;
"/="       return DE;
"="        return ASGN;
">"        return GT;
"<"        return LT;
"+"        return ADD;
"-"        return SUB;
"*"        return MUL;
"/"        return DIV;
"@"        return MOD;
"."        return DOT;
"&&"       return LA;
"||"       return LO;
"&"        return BA;
"|"        return BO;
"^"        return BXOR;
\/\/.*             ;
\/\*(.*\n)*.*\*\/    ;
. return yytext[0];
%%
```

Yacc Code (k.y):

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token INT FLOAT CHAR DOUBLE VOID LONG CONST TYPEDEF STRUCT
%token FOR DO WHILE BREAK CONTINUE
%token IF ELSE SWITCH CASE  DEFAULT
%token NUM ID MAIN
%token DOT RETURN
%left ADD SUB MUL DIV AE SE ME DE
%left LE GE EE NE LT GT LO LA BO BA BXOR
%right ASGN MOD "else"
%left '{'
%%
pgm   :   data_type MAIN '(' ')' block
           ;
block   :   '{' stmtlist '}'
             ;
stmtlist :  block
             |
             stmtlist stmt
             |
             ;
stmt    :   decl ';'
          |assignment ';'
          |ifelsestmt
          |whilestmt
          |switchstmt

          ;

decl   :    data_type { set_datatype(); } varlist //set_datatype() will store datatype in tmp

            ;

varlist :   varlist ',' ID { var_decl();}

          |

          ID { var_decl(); }   //var_decl store id in symbol table

          ;

assignment : ID { check_decl(); push();} assgn1  //check_decl() checks for redeclaration

                    ;

assgn1 :  ASGN {push();} expr {exp_asgn_evaluate();}          //evaluate expr

        |

        AE {push1("+");} expr{exp_evaluate(); } {exp_asgn_evaluate();} //shorthand operators
//grm rules

        |

        SE {push1("-");} expr{exp_evaluate(); } {exp_asgn_evaluate();}

        |

        ME {push1("*");} expr{exp_evaluate(); } {exp_asgn_evaluate();}

        |

        DE {push1("/");} expr{exp_evaluate(); } {exp_asgn_evaluate();}

         ;

ifelsestmt : IF '(' expr ')' { if_fun();} ifelsebody elsestmt  //if_fun checks for cond nt satisfied

                  ;

elsestmt   : ELSE {else_fun();} ifelsebody {if_end();}  //else_fun() to write goto of if end

               | {if_end();}

              ;

ifelsebody : block

           |

           stmt
           ;
whilestmt  :  {while_begin();} WHILE '(' expr ')' {while_body();} whilebody
                 ;  //while_begin() to generate a label for while at start
whilebody  : block {while_end();}    //while_end() to write goto to while_begin label
                 |
                 stmt {while_end();}
                 ;
switchstmt : SWITCH '(' expr ')' {switch_begin();} '{' switchbody '}'
                 ;
switchbody : switchcase {switch_end();}
                 ;
switchcase : CASE NUM {switchcase_fun();} ':' switcheval breakstmt
                 |
                 DEFAULT {switchdefault_fun();} ':' switcheval switchend
                 |
                 ;
breakstmt  : BREAK {break_fun();} ';' switchcase
                 |{no_break_fun();} switchcase
                 ;
switchend  : BREAK {break_fun();} ';'
                 |{no_break_fun();}
                 ;
switcheval : stmtlist
              ;
expr :    expr LO {push();} expr2 {exp_evaluate();}
        |
        expr2  //exp_evaluate() is to print 3 addr code as all values will be in stack
        ;
```

```
expr2 :   expr2 LA {push();} expr3 {exp_evaluate();}

        |

        expr3

        ;

expr3 :   expr3 BO {push();} expr4 {exp_evaluate();}

        |

        expr4

        ;

expr4 :   expr4 BXOR {push();}  expr5 {exp_evaluate();}

        |

        expr5

        ;

expr5 :   expr5 BA {push();} expr6 {exp_evaluate();}

            |

            expr6

            ;

expr6 :    expr6  LT {push();} expr7 {exp_evaluate();}

        |expr6  LE {push();} expr7 {exp_evaluate();}

        |expr6  GT {push();} expr7 {exp_evaluate();}

        |expr6  GE {push();} expr7 {exp_evaluate();}

        |expr6  NE {push();} expr7 {exp_evaluate();}

        |expr6  EE {push();} expr7 {exp_evaluate();}

        |expr7

        ;

expr7 :   expr7 ADD {push();} expr8 {exp_evaluate();}

        |expr7 SUB {push();} expr8 {exp_evaluate();}

        |expr8

        ;
```

```
expr8 :   expr8 MUL {push();} expr9 {exp_evaluate();}

         |expr8 DIV {push();} expr9 {exp_evaluate();}

         |expr9

         ;

expr9 :   expr10 MOD {push();} expr9 {exp_evaluate();}

         |

         expr10

         ;

expr10 : '(' expr ')'

        | ID  { check_decl();push();}

        | NUM { push(); }

        | SUB expr10 { unary_minus(); }  // for unary minus here itself once write 3 addr code
//nd then push tht operator on stack

        ;

data_type : INT

           |FLOAT

           |CHAR

           |DOUBLE

           |VOID

           |LONG

           |BOOL

           ;

%%

#include <ctype.h>

#include"lex.yy.c"

int ltrack[200][20];

int lcount[200]={0};

int lab_no[200];

int lineno=0;
```

```
struct store
{
int flag;
char op1[100];
int no;
} fetch[10000];
int count=0;
char st[1000][10];
int top=0;
int tmp_no=0;
char temp[8];
int lab_store[200];
int lnum=0;
int ltop=0;
int switch_store[1000];
int stop=0;
char dtype[10];
struct SymbolTable
{
        char id[20];
        char type[10];
}sym_store[10000];
int sym_count=0;
int main(int argc, char *argv[])
{
        yyin = fopen(argv[1], "r");
    if(!yyparse())
                printf("\nParsing complete\n");
```

```
            else
            {
                        printf("\nParsing failed\n");
                        exit(0);
            }
            fclose(yyin);
        outputcode();
        return 0;
}
yyerror(char *s)
{
            printf("Syntax Error In Line : %d : %s %s\n", yylineno, s, yytext );
}
push()
{
            strcpy(st[++top],yytext);
}
push1(char ch[])
{
   char ext[10];
   strcpy(st[++top],"=");
   strcpy(ext,st[top-1]);
   strcpy(st[++top],ext);
   strcpy(st[++top],ch);
}
unary_minus()
{
        sprintf(temp,"t%d",tmp_no);
```

```
            fetch[lineno].flag=0;
            strcpy(fetch[lineno].op1,temp);
            strcat(fetch[lineno].op1,"=");
            strcat(fetch[lineno].op1,"-");
            strcat(fetch[lineno].op1,st[top]);
            strcpy(st[top],temp);
            tmp_no++;
            lineno++;
}
exp_evaluate()
{
            sprintf(temp,"t%d",tmp_no);
            fetch[lineno].flag=0;
            strcpy(fetch[lineno].op1,temp);
            strcat(fetch[lineno].op1,"=");
            strcat(fetch[lineno].op1,st[top-2]);
            strcat(fetch[lineno].op1,st[top-1]);
            strcat(fetch[lineno].op1,st[top]);
            top-=2;
            strcpy(st[top],temp);
            tmp_no++;
            lineno++;
}
exp_asgn_evaluate()
{
        fetch[lineno].flag=0;
        strcpy(fetch[lineno].op1,st[top-2]);
            strcat(fetch[lineno].op1,"=");
            strcat(fetch[lineno].op1,st[top]);
            top-=3;
            lineno++;
}
if_fun()
{
            lnum++;
            fetch[lineno].flag=1;
            strcpy(fetch[lineno].op1,"if(not ");
            strcat(fetch[lineno].op1,st[top]);
            strcat(fetch[lineno].op1,") goto ");
            ltrack[lnum][lcount[lnum]++]=lineno;
            lab_store[++ltop]=lnum;
            lineno++;
}
else_fun()
{
            int x;
            lnum++;
            x=lab_store[ltop--];
            fetch[lineno].flag=1;
            strcpy(fetch[lineno].op1,"goto ");
            ltrack[lnum][lcount[lnum]++]=lineno;
            lineno++;
            int u=0;
            for(u=0;u<lcount[x];u++)
            fetch[ltrack[x][u]].no=lineno;
            lab_store[++ltop]=lnum;
```

```
}
if_end()
{
        int y;
        y=lab_store[ltop--];
        int u=0;
        for(u=0;u<lcount[y];u++)
        fetch[ltrack[y][u]].no=lineno;
        top--;
}
while_begin()
{
        lnum++;
        lab_store[++ltop]=lnum;
        lab_no[lnum]=lineno;
}
while_body()
{
        lnum++;
        fetch[lineno].flag=1;
        strcpy(fetch[lineno].op1,"if(not ");
        strcat(fetch[lineno].op1,st[top]);
        strcat(fetch[lineno].op1,") goto ");
        ltrack[lnum][lcount[lnum]++]=lineno;
        lab_store[++ltop]=lnum;
        lineno++;
}
while_end()
```

```
{
            int x,y;
            y=lab_store[ltop--];
            x=lab_store[ltop--];
        fetch[lineno].flag=1;
        strcpy(fetch[lineno].op1,"goto ");
        fetch[lineno].no=lab_no[x];
        lineno++;
            int u=0;
            for(u=0;u<lcount[y];u++)
            fetch[ltrack[y][u]].no=lineno;
            top--;
}
switch_begin()
{
            lnum++;
            lab_store[++ltop]=lnum;
            lnum++;
            lab_store[++ltop]=lnum;
            switch_store[++stop]=1;
}
switchcase_fun()
{
            int x,y,ck;
            ck=switch_store[stop--];
            if(ck==1)
            {
                        x=lab_store[ltop--];
            }
            else if(ck==2)
            {
                    y=lab_store[ltop--];
                    x=lab_store[ltop--];
            }
            int u=0;
            for(u=0;u<lcount[x];u++)
            fetch[ltrack[x][u]].no=lineno;
            lnum++;
            lab_store[++ltop]=lnum;
            fetch[lineno].flag=1;
            strcpy(fetch[lineno].op1,"if(");
            strcat(fetch[lineno].op1,st[top]);
            strcat(fetch[lineno].op1,"!=");
            strcat(fetch[lineno].op1,yytext);
            strcat(fetch[lineno].op1,") goto ");
            ltrack[lab_store[ltop]][lcount[lab_store[ltop]]++]=lineno;
        lineno++;
            if(ck==2)
            {
                    u=0;
                 for(u=0;u<lcount[y];u++)
                    fetch[ltrack[y][u]].no=lineno;
            }
    }
    switchdefault_fun()
    {
        int x,y,ck;
            ck=switch_store[stop--];
            if(ck==1)
            {
                        x=lab_store[ltop--];
            }
            else if(ck==2)
            {
                        y=lab_store[ltop--];
                        x=lab_store[ltop--];
            }
            int u=0;
            for(u=0;u<lcount[x];u++)
            fetch[ltrack[x][u]].no=lineno;
            lnum++;
            lab_store[++ltop]=lnum;
            if(ck==2)
            {
                        u=0;
                  for(u=0;u<lcount[y];u++)
                        fetch[ltrack[y][u]].no=lineno;
            }
}
break_fun()
{
            switch_store[++stop]=1;
        fetch[lineno].flag=1;
            strcpy(fetch[lineno].op1,"goto ");
            ltrack[lab_store[ltop-1]][lcount[lab_store[ltop-1]]++]=lineno;
            lineno++;
}
no_break_fun()
{
            switch_store[++stop]=2;
            lnum++;
            lab_store[++ltop]=lnum;
        fetch[lineno].flag=1;
            strcpy(fetch[lineno].op1,"goto ");
            ltrack[lab_store[ltop]][lcount[lab_store[ltop]]++]=lineno;
        lineno++;
}
switch_end()
{

        int x,y,ck;
            ck=switch_store[stop--];
            if(ck==1)
            {
                        x=lab_store[ltop--];
            }
            else if(ck==2)
            {
                        y=lab_store[ltop--];
                        x=lab_store[ltop--];
            }
            int u=0;
            for(u=0;u<lcount[x];u++)
            fetch[ltrack[x][u]].no=lineno;
            if(ck==2)
            {
                        u=0;
                     for(u=0;u<lcount[y];u++)
                        fetch[ltrack[y][u]].no=lineno;
            }
            x=lab_store[ltop--];
            u=0;
            for(u=0;u<lcount[x];u++)
            fetch[ltrack[x][u]].no=lineno;
            top--;
            stop--;
}
set_datatype()
{
            strcpy(dtype,yytext);
}
int is_predeclare(char temp[])
{
            int i;
            for(i=0;i<sym_count;i++)
            {
                        if(!strcmp(sym_store[i].id,temp))
                                return 1;
            }
            return 0;
```

```
}
var_decl()
{
        char temp[20];
        strcpy(temp,yytext);
        if(is_predeclare(temp))
        {
                yyerror("Redeclaration of variable ");
                exit(0);
        }
        else
        {
                strcpy(sym_store[sym_count].id,temp);
                strcpy(sym_store[sym_count].type,dtype);
                sym_count++;
        }
}
check_decl()
{
        char temp[20];
        strcpy(temp,yytext);
        int flag=0,i;
        for(i=0;i<sym_count;i++)
        {
                if(!strcmp(sym_store[i].id,temp))
                {
                   flag=1;
                   break;
```

```
                        }
            }
            if(!flag)
            {
                        yyerror("Not Predeclared variable ");
                        exit(0);
            }

}
outputcode()
{
int u=0;
            for(u=0;u<lcount[lineno];u++)
            {
            fetch[ltrack[lineno][u]].no=lineno;
            }
for(u=0;u<lineno;u++)
{
if(fetch[u].flag==0)
printf("%d: %s\n",u,fetch[u].op1);
else
{
printf("%d: %s ",u,fetch[u].op1);
printf("%d\n",fetch[u].no);
}
}
printf("%d: PGM END\n",lineno);
}
```

Examples:-

1) Input file (file1)

```
 void main()
{
      int a,b,c,d,s,r,p,q,h,w,res;
       w=p||q&&h;
       if(a>b)
       {
         b=a+c;
       }
       if(b==a)
       {
       s=r;
       }
       else
       {
        if(p>q)
          w=s;
        else
        h=b;
       }
      while(a+b*c)
      {
              if(a||b&&c)
              a=b;
              else
              c=d;
              d=p*q+c*(a+b);
```

```
                q=r;
        }
        while(b+d)
        {
        switch(a+b)
        {
        case 1: b=c;
        break;
        case 2: s=p*q;
        break;
        case 3: p=q;
        //break;
        default: a=b+c;
        //break;
        }
        }
        a=b@c@d;
        a=c+-(b+c@d*c);
}
```

Output:-

```
0: t0=q&&h
1: t1=p||t0
2: w=t1
3: t2=a>b
4: if(not t2) goto  7
5: t3=a+c
6: b=t3
7: t4=b==a
```

```
8: if(not t4) goto  11

9: s=r

10: goto  16

11: t5=p>q

12: if(not t5) goto  15

13: w=s

14: goto  16

15: h=b

16: t6=b*c

17: t7=a+t6

18: if(not t7) goto  32

19: t8=b&&c

20: t9=a||t8

21: if(not t9) goto  24

22: a=b

23: goto  25

24: c=d

25: t10=p*q

26: t11=a+b

27: t12=c*t11

28: t13=t10+t12

29: d=t13

30: q=r

31: goto  16

32: t14=b+d

33: if(not t14) goto  49

34: t15=a+b

35: if(t15!=1) goto  38
```

```
36: b=c
37: goto  48
38: if(t15!=2) goto  42
39: t16=p*q
40: s=t16
41: goto  48
42: if(t15!=3) goto  45
43: p=q
44: goto  45
45: t17=b+c
46: a=t17
47: goto  48
48: goto  32
49: t18=c@d
50: t19=b@t18
51: a=t19
52: t20=c@d
53: t21=t20*c
54: t22=b+t21
55: t23=-t22
56: t24=c+t23
57: a=t24
58:PGM END
```

2) Input file (file2)

```
 int main()
{
  int a,b,c,d;
```

```
if(a+b)
 b=-c;
else
{
switch(a+b)
  {
        case 0: d=a+b;
        break;
        case 1:
        {
              a=b;
              switch(b+-4)
              {
                    case 10: d=b@c@d*a;
                    break;
                    case 20:
                    case 30: d=c;
                    break;
                    default: b=c;
                    break;
              }
        }
        case 2: a=-(b*c+d);
        break;
        case 3: b=b+d;
        case 4: c=c&&d&&a||b;
        case 5: b*=-c;
        default: a=b;
```

```
              break;
      }
    }


  a/=b;
  b+=-c+(a&&b^c)/d;
}
```

Output:

```
0: t0=a+b
1: if(not t0) goto  5
2: t1=-c
3: b=t1
4: goto  51
5: t2=a+b
6: if(t2!=0) goto  10
7: t3=a+b
8: d=t3
9: goto  51
10: if(t2!=1) goto  28
11: a=b
12: t4=-4
13: t5=b+t4
14: if(t5!=10) goto  20
15: t6=c@d
16: t7=b@t6
17: t8=t7*a
18: d=t8
```

```
19: goto  27
20: if(t5!=20) goto  22
21: goto  23
22: if(t5!=30) goto  25
23: d=c
24: goto  27
25: b=c
26: goto  27
27: goto  29
28: if(t2!=2) goto  34
29: t9=b*c
30: t10=t9+d
31: t11=-t10
32: a=t11
33: goto  51
34: if(t2!=3) goto  38
35: t12=b+d
36: b=t12
37: goto  39
38: if(t2!=4) goto  44
39: t13=c&&d
40: t14=t13&&a
41: t15=t14||b
42: c=t15
43: goto  45
44: if(t2!=5) goto  49
45: t16=-c
46: t17=b*t16
```

```
47: b=t17

48: goto  49

49: a=b

50: goto  51

51: t18=a/b

52: a=t18

53: t19=-c

54: t20=b^c

55: t21=a&&t20

56: t22=t21/d

57: t23=t19+t22

58: t24=b+t23

59: b=t24

60:PGM END
```