CS6612 – Compiler Lab

Ex no : 7 **Name : Sreedhar V**

Date : 23.04.2021 **Reg no: 185001161**

Programming Assignment-7 - Generation of Intermediate Code using Lex and Yacc

---

The new Language **Pascal-2021** is introduced with the following programming constructs

Data types

integer

real

char

Operators

+, -, * and /

Precedence → * and / have lesser priority than + and –

Associativity → * and / → right , + and - → left

Declaration statement

var: type;

var: type=constant;

Example

a: integer;

b: integer = 5;

Generate Intermediate code (TAC sequences) for the code involving conditional and assignment statements.

Conditional Statement

if condition then

---

else

---

end if

Generate Intermediate code in the form of Three Address Code sequence for the sample input program written using declaration, conditional and assignment statements in new language **Pascal-2021**, Following is the sample input

Code

Lex file(tac.l)

```
%{
struct info{
        char *var;
        char *code;
        int val;
    };
#include <stdio.h>
#include<string.h>
#include "tac.tab.h"
void yyerror(char *);
extern YYSTYPE yylval;
%}
id ([a-zA-Z_][a-zA-Z0-9_]*|[0-9]+)
num [0-9]+
GT ">="
LT "<="
LS "<<"
RS ">>"
LAND "&&"
LOR "||"
EQ "=="
NQ "!="
CHAR ['][a-zA-Z0-9]['] 
%%
begin {return BEG;}
end {return END;}
if {return IF;}
then {return THEN;}
else {return ELSE;}
end_if {return ENDIF;}
integer {return INT;}
real {return REAL;}
char {return CHAR;}
var {return VAR;}
{num} {yylval.temp.val=atoi(yytext);return NUM;}
{id} {yylval.temp.var=(char*)malloc(10);strcpy(yylval.temp.var,yytext);
return ID;}
{GT} {return GT;}
{LT} {return LT;}
{LS} {return LS;}
```

```
{RS} {return RS;}
{LAND} {return LAND;}
{LOR} {return LOR;}
{NQ} {return NQ;}
{EQ} {return EQ;}
[{};()]  {return *yytext;}
[-+*/^()=&|%:;] {return *yytext;}
{CHAR} {yylval.temp.var=(char*)malloc(10);strcpy(yylval.temp.var,yytext
);return CH;}
[\t] ;
[\n] ;
[ ] ;

%%
int yywrap(void){return 1;}
```

yacc file(tac.y)

```
%{
#include <stdio.h>

#include <math.h>
#include<stdlib.h>
#include<string.h>
struct table{
    char var[20];
    int val;
    char type[20];
}symbol[10];
int l=0,t=1,n=0;
int yyerror(char *er);
int yylex(void);
void display_table()
{
    int j=0;
    printf("\tSYMBOL TABLE\n");
    printf("Name      Type      Value\n");
    for(j=0;j<n;j++)
    {
        printf("%-10s %-10s %-
10d\n",symbol[j].var,symbol[j].type,symbol[j].val);
```

```
    }
}
struct info{
        char *var;
        char *code;
        int val;
};

%}
%token NUM LS RS GT LT LAND EQ NQ LOR ID IF THEN BEG END ELSE INT CHAR
REAL CH ENDIF VAR

%union{

    struct info temp;
    int val;
    char *code;
}

%type<code> S BLOCK ASSIGNMENT CONDITION
%type<temp> E C ID
%type<val> NUM

%right '='
%left '!'
%left LOR
%left LAND
%left '|'
%left '&'
%left EQ NQ
%left '>' GT
%left '<' LT
%left LS RS
%right '*' '^' '/' '%'
%left '+' '-'
%left '(' ')'

%%
S : DEC BEG BLOCK END {printf("\nBEGIN %s\n END\n Syntactically Correct
\n",&$3);display_table();return 0;}
DEC   :DEC DEC
      | VAR ID ':' INT '=' NUM ';' {strcpy(symbol[n].var,$2.var);strcpy
(symbol[n].type,"INT");symbol[n++].val=$6;}
      | VAR ID ':' REAL '=' NUM ';' {strcpy(symbol[n].var,$2.var);strcp
y(symbol[n].type,"REAL");symbol[n++].val=$6;}
```

```
      | VAR ID ':' REAL ';'         {strcpy(symbol[n].var,$2.var);strcp
y(symbol[n].type,"REAL");symbol[n++].val=0;}
      | VAR ID ':' INT ';'          {strcpy(symbol[n].var,$2.var);strcp
y(symbol[n].type,"INT");symbol[n++].val=0;}
      | VAR ID ':' CHAR ';'         {strcpy(symbol[n].var,$2.var);strcp
y(symbol[n].type,"CHAR");symbol[n++].val=0;}

BLOCK : CONDITION {$$=(char*)malloc(2000);sprintf($$,"%s\n",$1);}
      | ASSIGNMENT';' {$$=(char*)malloc(2000);sprintf($$,"%s\n",$1);}
      | BLOCK BLOCK {$$=(char*)malloc(2000);sprintf($$,"%s%s\n",$1,$2);
}
      | {$$=(char*)malloc(2000);sprintf($$,"");}

ASSIGNMENT : ID '=' E {$$=(char*)malloc(2000);sprintf($$,"%s %s=%s\n",$
3.code,$1.var,$3.var);}
           | ID '+''+' {$$=(char*)malloc(2000);sprintf($$,"%s++\n",$1.v
ar);}
           | ID '-''-' {$$=(char*)malloc(2000);sprintf($$,"%s--
\n",$1.var);}


CONDITION : IF '(' C ')' THEN BLOCK ELSE BLOCK ENDIF {$$=(char*)malloc(
2000);sprintf($$,"   if %s goto L%d\n   goto L%d\nL%d:\n%s   goto L%d\n
L%d:\n%sL%d:\n",$3.code,l,l+1,l,$6,l+2,l+1,$8,l+2);l+=3;}
          | IF '(' C ')' THEN BLOCK ENDIF {$$=(char*)malloc(2000);sprin
tf($$,"   if %s goto L%d\n   goto L%d\nL%d:\n%sL%d:\n",$3.code,l,l+1,l,
$6,l+1);l+=2;}

E : NUM {$$.var=(char*)malloc(3);sprintf($$.var,"%d",$1);$$.code=(char*
)malloc(1);strcpy($$.code,"");}
  | E '+' E {$$.var=(char*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.co
de=(char*)malloc(300);sprintf($$.code,"%s%s   %s = %s + %s\n",$1.code,$
3.code,$$.var,$1.var,$3.var);}
  | E '-
' E {$$.var=(char*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.code=(char
*)malloc(300);$$.code=(char*)malloc(300);sprintf($$.code,"%s%s   %s = %
s - %s\n",$1.code,$3.code,$$.var,$1.var,$3.var);}
  | E '*' E {$$.var=(char*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.co
de=(char*)malloc(300);sprintf($$.code,"%s%s   %s = %s * %s\n",$1.code,$
3.code,$$.var,$1.var,$3.var);}
  | E '/' E {$$.var=(char*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.co
de=(char*)malloc(300);sprintf($$.code,"%s%s   %s = %s / %s\n",$1.code,$
3.code,$$.var,$1.var,$3.var);}
  | E '%' E {$$.var=(char*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.co
de=(char*)malloc(300);sprintf($$.code,"%s%s   %s = %s % %s\n",$1.code,$
3.code,$$.var,$1.var,$3.var);}
```

```
  | E '^' E {$$.var=(char*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.co
de=(char*)malloc(300);sprintf($$.code,"%s%s   %s = %s ^ %s\n",$1.code,$
3.code,$$.var,$1.var,$3.var);}
  | E RS E {$$.var=(char*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.cod
e=(char*)malloc(300);sprintf($$.code,"%s%s   %s = %s >> %s\n",$1.code,$
3.code,$$.var,$1.var,$3.var);}
  | E LS E {$$.var=(char*)malloc(3);sprintf($$.var,"t%d",t);t+=1;$$.cod
e=(char*)malloc(300);sprintf($$.code,"%s%s   %s = %s << %s\n",$1.code,$
3.code,$$.var,$1.var,$3.var);}
  | '(' E ')' {$$.var=(char*)malloc(3);sprintf($$.var,"%s",$2.var);$$.c
ode=(char*)malloc(300);sprintf($$.code,"%s\n",$2.code);}
  | ID  {$$.var=(char*)malloc(3);sprintf($$.var,"%s",$1.var);$$.code=(c
har*)malloc(300);strcpy($$.code,"");}

C : E GT E {$$.code=(char*)malloc(300);sprintf($$.code,"%s >= %s",$1.va
r,$3.var);}
    | E '>' E {$$.code=(char*)malloc(300);sprintf($$.code,"%s > %s",$1.
var,$3.var);}
    | E '<' E {$$.code=(char*)malloc(300);sprintf($$.code,"%s < %s",$1.
var,$3.var);}
    | E LT E {$$.code=(char*)malloc(300);sprintf($$.code,"%s <= %s",$1.
var,$3.var);}
    | E '&' E {$$.code=(char*)malloc(300);sprintf($$.code,"%s & %s",$1.
var,$3.var);}
    | E '|' E {$$.code=(char*)malloc(300);sprintf($$.code,"%s | %s",$1.
var,$3.var);}
    | E LAND E {$$.code=(char*)malloc(300);sprintf($$.code,"%s && %s",$
1.var,$3.var);}
    | E LOR E {$$.code=(char*)malloc(300);sprintf($$.code,"%s || %s",$1
.var,$3.var);}
    | '!' E {$$.code=(char*)malloc(300);sprintf($$.code,"! %s",$2.var);
}
    | E EQ E {$$.code=(char*)malloc(300);sprintf($$.code,"%s == %s",$1.
var,$3.var);}
    | E NQ E {$$.code=(char*)malloc(300);sprintf($$.code,"%s != %s",$1.
var,$3.var);}


%%
int main()
{
    printf("\n\n\nIntermediate code generation PASCAL-
2021 language\n");
    yyparse();
}
int yyerror(char *er)
```

```
{
    printf("\nInvalid character %s\n",er);
    exit(0);
}
```

Learning Outcome:

- I’ve learnt how to implement the syntax checker considering all its grammar rules , operator precedence and syntax for a custom language while execution.
- I’ve learnt the basic syntax of the yacc program and how to implement the grammar rules in c code.
- I’ve learnt how to give use different datatypes for the top of the stack using union.
- I’ve leant how the lex program sends the token based on its syntax and yacc program evaluates the stream of tokens based on the given grammar rules and produces the result.