Assignment-7

Generation of Intermediate Code using Lex and Yacc

Name: Shashank R
Roll No:185001144

AIM:

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**,

CODE:

Yacc Program:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define YYSTYPE struct info*
int err_flag = 0;

struct info {
        int no;
        char* var;
        char* code;
};

typedef struct info info;
info temp;

int cnt_token = 0, cnt_label = 0;

info* getNewNode(char* v){
        //cnt_token++;
        info *ret = (info*)malloc(sizeof(info) * 1);
        ret->code = (char*)malloc(sizeof(char) * 100);
        ret->var = (char*)malloc(sizeof(char) * 100);
        ret->var[0] ='\0';
        strcpy(ret->code, v);
        ret->no = cnt_token - 1;
        return ret;
}

int count = 0;
```

```
%}


%token ID DT VAR
%token ARITHOP RELOP LOGOP
%token CHARCONST STRCONST NUMCONST
%token IF THEN ELSE ENDIF
%left LOGOP
%left RELOP
%left ARITHOP
%left '!'

%%
Program         :       Block
                        {
                                printf("%s", $$->code);
                        }
                ;

Block           :       Line Block
                        {
                                char cur[1]; cur[0] = '\0';
                                info* t  = getNewNode(cur);
                                sprintf(t->code, "%s\n%s", $1->code, $2->code);
                                $$ = t;
                        }
                |       Line
                        {
                                $$ = $1;
                        }
                ;

Line            :       IF '(' Expr ')' THEN Block ELSE Block ENDIF
                        {
                                char cur1[3];
                                cur1[0] = 'L'; cur1[1] = cnt_label + '0'; cur1[2] = '\0';
                                cnt_label++;

                                char cur2[3];
                                cur2[0] = 'L'; cur2[1] = cnt_label + '0'; cur2[2] = '\0';
                                cnt_label++;

                                char cur[1];
                                cur[0] = '\0';
                                info* t  = getNewNode(cur);

                                sprintf(t->code, "\tif %s goto %s \n\tgoto
%s\n\n%s:%s\n%s:%s\n", $3->code, cur1, cur2, cur1, $6->code, cur2, $8->code);
                                $$ = t;
```

```
		}
	|	IF '(' Expr ')' THEN Block
	|	Expr';'
		{
			char cur[1];
			cur[0] = '\0';
			info* t  = getNewNode(cur);
			sprintf(t->code, "\t%s\n", $1->code);
			$$ = t;
		}
	|	Decl';'
	;


Expr		:	Assign
	|	Expr ARITHOP'=' Expr
	|	Expr ARITHOP Expr
		{
			info* lt = $1;
			info* rt = $3;
			char cur[1];
			cur[0] = '\0';
			info* t  = getNewNode(cur);
			char cur1[10];
			cur1[0] = 't'; cur1[1] = cnt_token + '0'; cur1[2] = '\0';
			cnt_token++;

			if(strlen(lt->var) > 0){
				sprintf(t->var, "%s", cur1);
				sprintf(t->code, "%s\n\t%s = %s %c %s", lt->code,
cur1, lt->var, $2, rt->code);
			}else{
				sprintf(t->var, "%s", cur1);
				sprintf(t->code, "%s = %s %c %s", cur1, lt->code,
$2, rt->code);
			}

			$$ = t;
		}
	|	Expr RELOP Expr
		{
			info* lt = $1;
			info* rt = $3;
			char cur[1]; cur[0] = '\0';
			info* t  = getNewNode(cur);
			sprintf(t->code, "%s %c %s", lt->code, $2, rt->code);
			$$ = t;
		}
	|	Expr LOGOP Expr
```

```
                        |       ID
                                {

                                        char cur[10];
                                        cur[0] = $1; cur[1] = '\0';
                                        info* t  = getNewNode(cur);
                                        $$ = t;
                                }
                        |       NUMCONST
                        |       '!'Expr
                        ;


Decl            :       VAR':'DT Assign
                        ;

Assign          :       ID'='CHARCONST
                        |       ID'='STRCONST
                        |       ID'='Expr
                                {
                                        info* lt = $1;
                                        info* rt = $3;
                                        char cur[10];
                                        cur[0] = 't'; cur[1] = cnt_token + '0'; cur[2] = '\0';
                                        cnt_token++;
                                        info* t  = getNewNode(cur);
                                        if(strlen(rt->var) > 0){
                                                sprintf(t->code, "%s\n\t%s = %c", rt->code, rt->var,
$1);
                                        }else{
                                                sprintf(t->code, "%s = %c", rt->code, $1);
                                        }
                                        $$ = t;
                                }
                        ;

%%



int yyerror(){
        err_flag = 1;
        return 1;
}

int main(){
   printf("\n\t\tSYNTAX CHECKER USING YACC\n");
   yyparse();
   return 1;
```

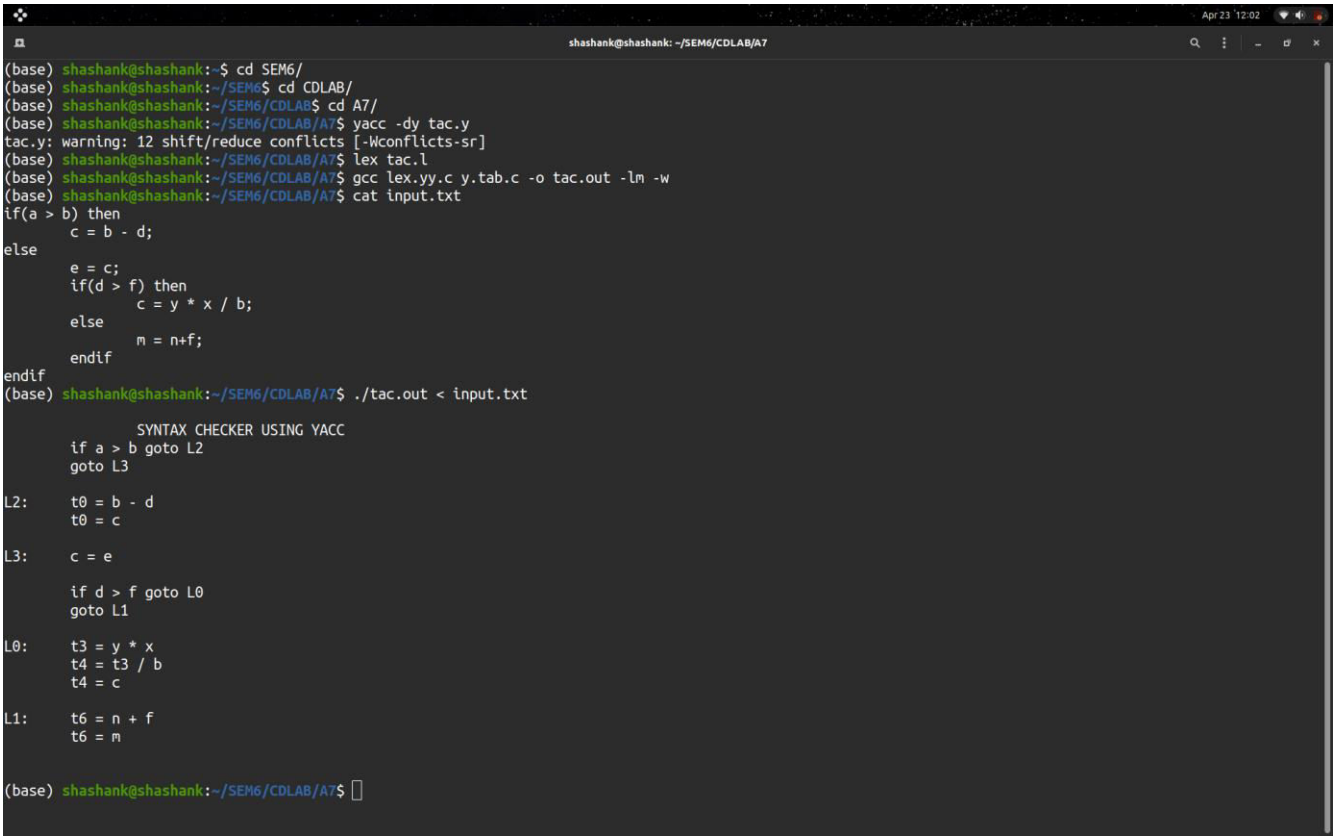
```
}
lex Program

%option noyywrap
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
extern int yylval;

%}

datatype    "int"|"real"|"char"
var         "var"
if          "IF"|"if"
then        "THEN"|"then"
endif       "ENDIF"|"endif"
else        "ELSE"|"else"

letter      [a-zA-Z]
letters     {letter}+
digit       [0-9]
digits      [+-]?{digit}+
optfrac     [.]{digits}
optexp      [E][+-]?{digits}
numconst    {digits}({optfrac}|{optexp})?
charconst   [']{letter}[']
strconst    ["]{letters}["]

id          {letter}({letters}|{digits})?
arithop     ("+"|"-"|"*"|"/"|"%")
relop       ("<="|">="|">"|"<"|"=="|"!=")
logop       ("&&"|"||"|"!")
newl        [\n]
tabs        [\t]
spaces      [ ]

%%
{var}           { return VAR;}
{if}            { return IF;}
{then}          { return THEN;}
{else}          { return ELSE;}
{endif}         { return ENDIF;}
{datatype}      { return DT;}
{numconst}      { yylval = atoi(yytext); return NUMCONST;}
{charconst}     { return CHARCONST;}
{strconst}      { return STRCONST;}
{id}            { yylval = yytext[0]; return ID;}
{arithop}       { yylval = yytext[0]; return ARITHOP;}
```

```
{relop}         { yylval = yytext[0]; return RELOP;}
{logop}         { return LOGOP;}
{spaces}        {};
{newl}          {};
{tabs}          {};
.               {return *yytext;};
%%
```

input.txt

```
if(a > b) then
        c = b - d;
else
        e = c;
        if(d > f) then
                c = y * x / b;
        else
                m = n+f;
        endif
endif
```

Screenshots


Apr 23 12:02
shashank@shashank: ~/SEM6/CDLAB/A7
(base) shashank@shashank:~$ cd SEM6/
(base) shashank@shashank:~/SEM6$ cd CDLAB/
(base) shashank@shashank:~/SEM6/CDLAB$ cd A7/
(base) shashank@shashank:~/SEM6/CDLAB/A7$ yacc -dy tac.y
tac.y: warning: 12 shift/reduce conflicts [-Wconflicts-sr]
(base) shashank@shashank:~/SEM6/CDLAB/A7$ lex tac.l
(base) shashank@shashank:~/SEM6/CDLAB/A7$ gcc lex.yy.c y.tab.c -o tac.out -lm -w
(base) shashank@shashank:~/SEM6/CDLAB/A7$ cat input.txt
if(a > b) then
c = b - d;
else
e = c;
if(d > f) then
c = y * x / b;
else
m = n+f;
endif
endif
(base) shashank@shashank:~/SEM6/CDLAB/A7$ ./tac.out < input.txt
SYNTAX CHECKER USING YACC
if a > b goto L2
goto L3
L2: t0 = b - d
t0 = c
L3: c = e
if d > f goto L0
goto L1
L0: t3 = y * x
t4 = t3 / b
t4 = c
L1: t6 = n + f
t6 = m
(base) shashank@shashank:~/SEM6/CDLAB/A7$