## Compiler Design Week-1

**WEEK-1:** Design a Lexical analyzer for C language. The lexical analyzer should ignore redundant spaces, tabs and newlines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value.

1. **Is constant or not**

#include<stdio.h> #include<stdlib.h> int main(){

char s[20]; int val;

printf("Enter the string:\n"); gets(s);

val = atoi(s); if (val){

printf("The given string is constant\n");

}

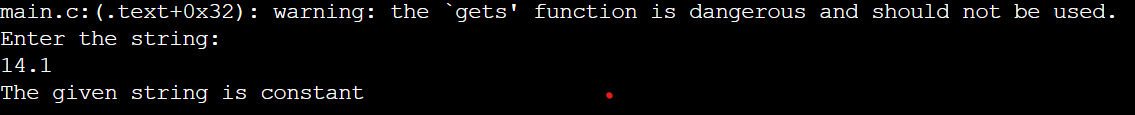
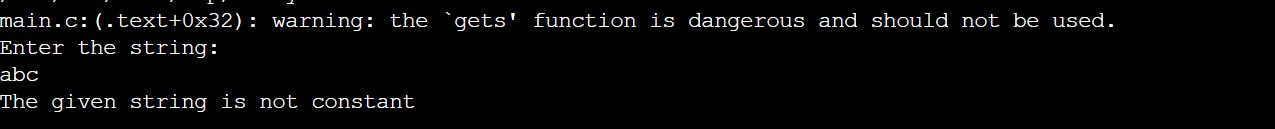
else{

}

printf("The given string is not constant\n");

eturn 0;

}



##### Checking comment lines

#include<stdio.h> #include<string.h> int main(){

char s[50]; int n;

printf("Enter the string:\n"); gets(s);

if (s[0] == '/'){

if (s[1] == '/'){

printf("Given statement is a comment\n");

}

else if (s[1] == '\*'){

n = strlen(s) - 1;

if (s[n] == '/' && s[n-1] == '\*'){

printf("Given statement is a comment\n");

}

else{

}

else{

}

printf("Given statement is not a comment\n");

}

}

else{

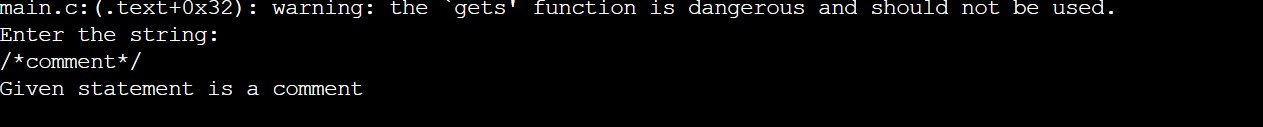
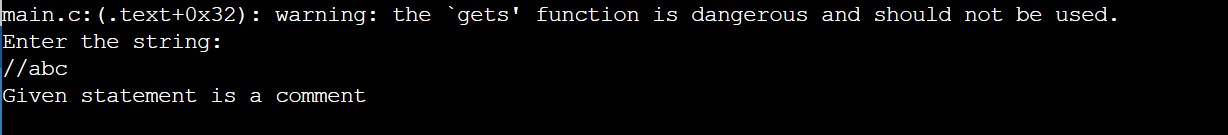
printf("Given statement is not a comment\n");

rintf("Given statement is not a comment\n");

}

return 0;

}



##### Checking identifies

#include<stdio.h> #include<string.h>

int main(){ char s[50]; int flag = 0,i;

printf("Enter the string:\n"); gets(s);

if (isalpha(s[0]) || s[0] == '\_'){ for(i=1;i<strlen(s);i++){

if (isdigit(s[i]) || isalpha(s[i]) || s[i] == '\_'){ flag = 1;

}

else{

}

}

}

break;

if (flag == 1){

printf("Given string is valid identifier\n");

}

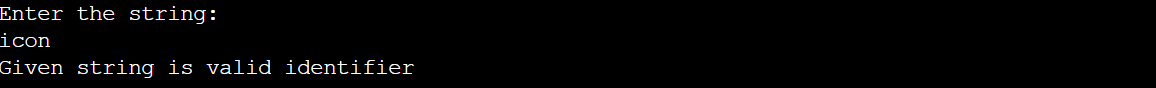
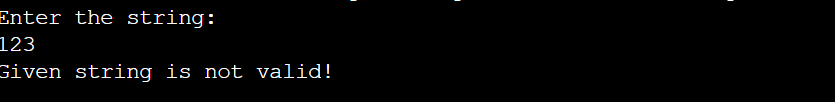
else{

}

printf("Given string is not valid!\n");

return 0;

}



##### checking keywords

#include<stdio.h>

#include<string.h> int main(){

char s[50];

char token[20][10];

int j=0,k=0,i,cnt = 0,flag = 0; char

keys[21][10]={"auto","double","struct","break","else","long","switch","case","enum","register","typ

e def","char","extern","return","for","const","float","short","do","if","while"}; gets(s);

for(i=0;i<strlen(s);i++){ if (s[i] != ' '){

token[j][k] = s[i]; k += 1;

}

else if (s[i] == ' '){

token[j][k] = '\0'; j += 1;

k = 0;

cnt += 1;

}

}

token[j][k] = '\0'; cnt += 1;

for (i=0;i<cnt;i++){

for(j = 0;j<21;j++){

if(strcmp(token[i],keys[j]) == 0){

printf("%s is a keyword\n",token[i]); flag = 1;

}

}

}

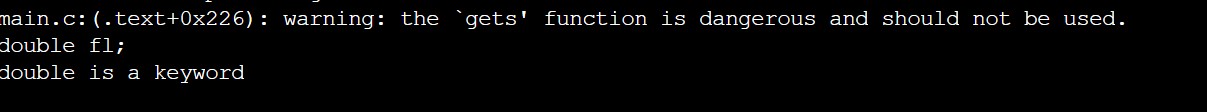
if (flag == 0){

printf("No keywords in given statement\n");

}

return 0;

}



##### Checking operators

#include<stdio.h> int main(){

char s[5];

printf("Enter the operator:\n"); gets(s);

switch(s[0]){

case '>':{

if (s[1] == '=')

printf("Greater than Equal to\n");

else

break;} case '<':{

printf("Greater than\n");

if (s[1] == '=')

printf("Less than Equal to\n");

else

break;} case '=':{

printf("Less than\n");

if (s[1] == '=')

printf("Comparing operator\n");

else

break;} case '|':{

printf("Assignment operator\n");

if (s[1] == '|')

printf("Logical OR\n");

else

break;} case '&':{

printf("Bitwise OR\n");

if (s[1] == '&')

else break;}

printf("Logical AND\n"); printf("Bitwises AND\n");

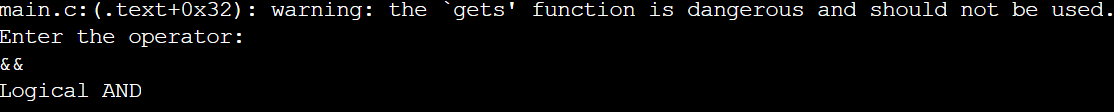
case '+':{printf("Addition operator\n");break;} case '-':{printf("Subtraction operator\n");break;} case '\*':{printf("Multiplication operator\n");break;} case '/':{printf("Division operator\n");break;} case '%':{printf("Modulo operator\n");break;} default:

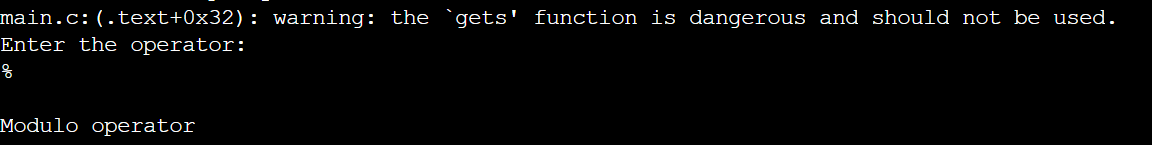
printf("Not an operator\n");

}

return 0;

}





## CD\_Week2

### Implement the lexical analyzer using lex.

#### Week2.l

%{

#include<stdio.h>

%}

digit [0-9]+ word [A-Za-z]+

spsym [(){};,%\{\]] arith [+-/\*] whitspc[ \t\n] underscr[\_]

%%

{whitspc}+ ;

\"[^\n\"]\*\" {printf("\n %s is a literal",yytext);} int |

include | if |

else | while | do | switch | case | default | break |

continue |

scanf {printf("\n%s is a Keyword",yytext);}

{spsym} {printf("\n%s is a Special Symbol",yytext);}

{arith} {printf("\n%s is a Binary Operator",yytext);}

= {printf("\n%s is a Assignment operator",yytext);} "++" |"--" {printf("\n%s is an Unary Operator",yytext);}

"&" |"|" |"^" {printf("\n %s is bitwise operator",yytext);} "<" |">" |"<=" |">=" |"==" |"!=" {printf("\n %s is a relational operator",yytext);}

{digit}+ {printf("\n %s is an integer constant",yytext);} ({digit}+)|({digit}\*\.{digit}+) {printf("\n %s is an float constant",yytext);} ({word}({word}|{digit}|{underscr})\*) {printf("\n%s is a Identifier",yytext);}

%%

int main(int argc,char \*argv[])

{

FILE \*fp; fp=fopen(argv[1],"r"); if(!fp)

{

printf("cnt open:%s",argv[1]); exit(1);

}

yyin=fp; yylex();

}

int yywrap()

{

return 1;

}

#### f1.c:

//var.c #include<stdio.h> #include<conio.h> void main()

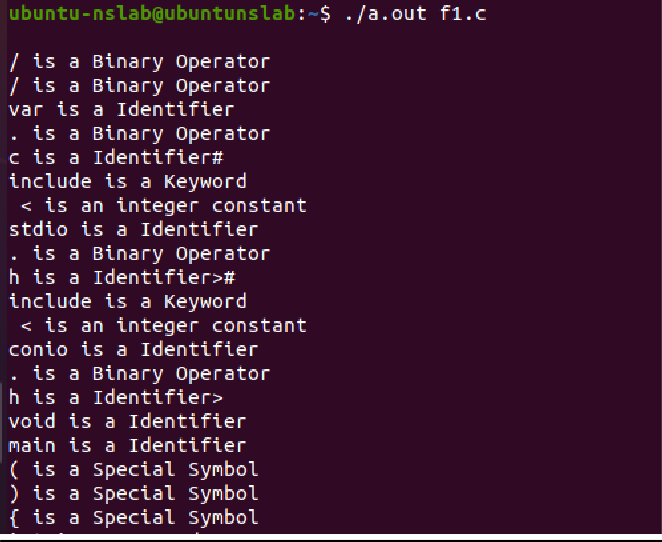
{

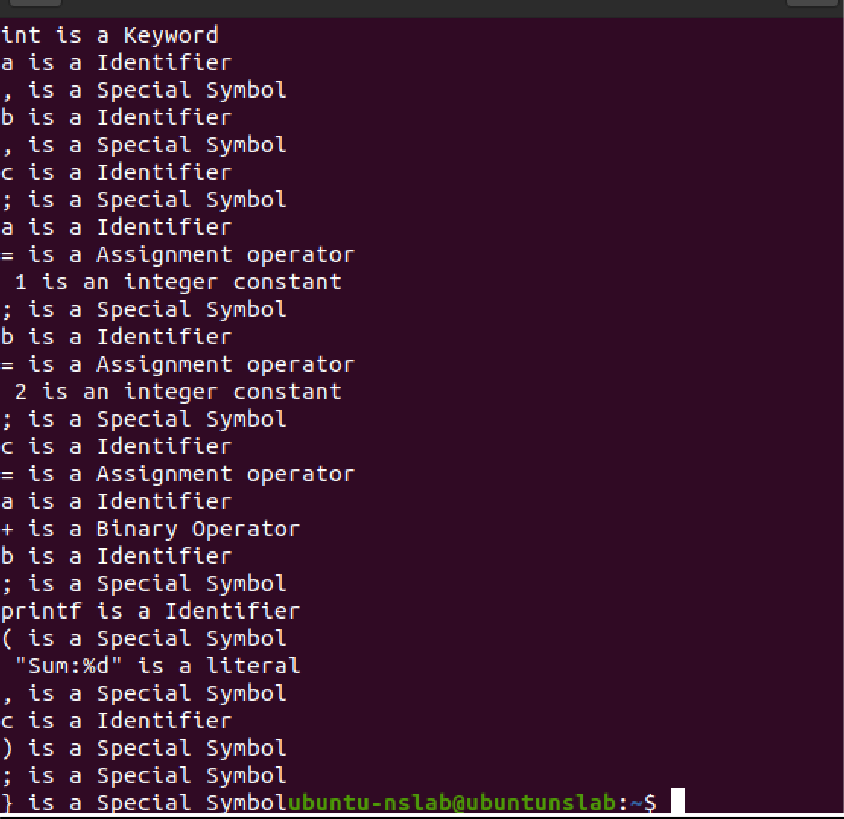
int a,b,c; a=1; b=2;

c=a+b; printf("Sum:%d",c);

}

#### OUTPUT:





**CD-Week3**

### 1. Design Predictive parser for a given language.

#include <stdio.h> #include <string.h>

char prol[7][10] = { "S", "A", "A", "B", "B", "C", "C" };

char pror[7][10] = { "A", "Bb", "Cd", "aB", "@", "Cc", "@" };

char prod[7][10] = { "S->A", "A->Bb", "A->Cd", "B->aB", "B->@", "C->Cc", "C->@" };

char first[7][10] = { "abcd", "ab", "cd", "a@", "@", "c@", "@" };

char follow[7][10] = { "$", "$", "$", "a$", "b$", "c$", "d$" }; char table[5][6][10];

int numr(char c)

{

switch (c)

{

case 'S': return 0; case 'A': return 1; case 'B': return 2; case 'C': return 3; case 'a': return 0; case 'b': return 1; case 'c': return 2; case 'd': return 3; case '$': return 4;

}

return (2);

}

int main()

{

int i, j, k;

for (i = 0; i < 5; i++)

for (j = 0; j < 6; j++) strcpy(table[i][j], " ");

printf("The following grammar is used for Parsing Table:\n");

for (i = 0; i < 7; i++) printf("%s\n", prod[i]);

printf("\nPredictive parsing table:\n"); fflush(stdin);

for (i = 0; i < 7; i++)

{

k = strlen(first[i]); for (j = 0; j < 10; j++) if (first[i][j] != '@')

strcpy(table[numr(prol[i][0]) + 1][numr(first[i][j]) + 1], prod[i]);

}

for (i = 0; i < 7; i++)

{

if (strlen(pror[i]) == 1)

{

if (pror[i][0] == '@')

{

k = strlen(follow[i]); for (j = 0; j < k; j++)

strcpy(table[numr(prol[i][0]) + 1][numr(follow[i][j]) + 1], prod[i]);

}

}

}

strcpy(table[0][0], " ");

strcpy(table[0][1], "a");

strcpy(table[0][2], "b");

strcpy(table[0][3], "c");

strcpy(table[0][4], "d");

strcpy(table[0][5], "$");

strcpy(table[1][0], "S");

strcpy(table[2][0], "A");

strcpy(table[3][0], "B");

strcpy(table[4][0], "C");

printf("\n \n"); for (i = 0; i < 5; i++)

for (j = 0; j < 6; j++)

{

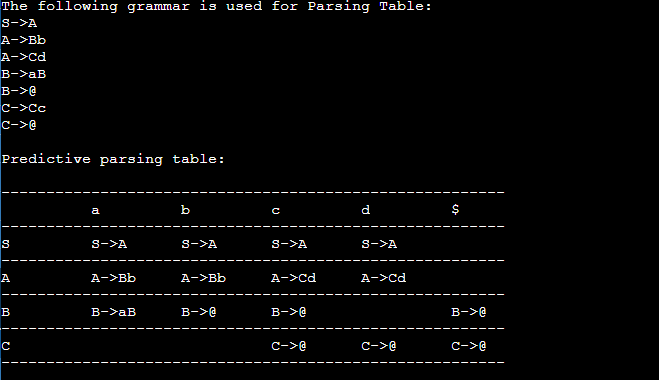
printf("%-10s", table[i][j]); if (j == 5)

printf("\n \n");

}

}

**OUTPUT:**



## CD\_Week4

### week4.l:

%{

#include<stdio.h> #include "y.tab.h"

%}

%%

[0-9]+ {yylval.dval=atof(yytext); return DIGIT;

}

\n|. return yytext[0];

%%

### week4.y:

%{

/\*This YACC specification file generates the LALR parser for the program considered in experiment 4.\*/

#include<stdio.h>

%}

%union

{

double dval;

}

%token <dval> DIGIT

%type <dval> expr

%type <dval> term

%type <dval> factor

%%

line: expr '\n' { printf("%g\n",$1);

}

;

expr: expr '+' term {$$=$1 + $3 ;}

| term

;

term: term '\*' factor {$$=$1 \* $3 ;}

| factor

;

factor: '(' expr ')' {$$=$2 ;}

| DIGIT

;

%%

int main()

{

yyparse();

}

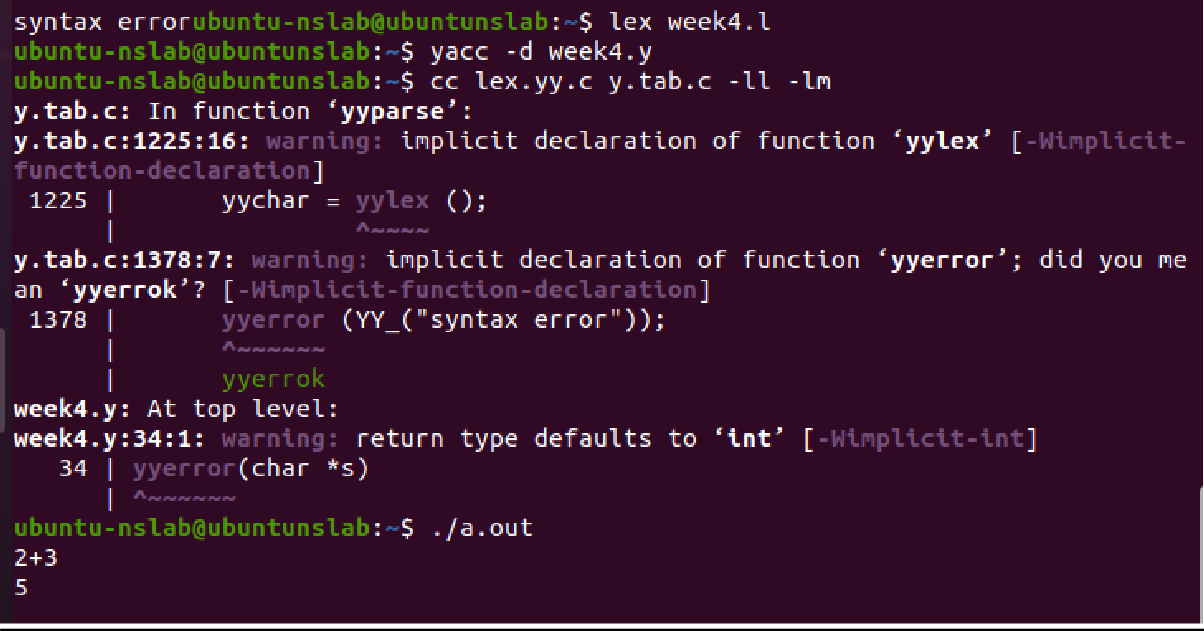
yyerror(char \*s)

{

printf("%s",s);

}

**OUTPUT:**



# Week 5:

### Convert the BNF rules into YACC form and write code to generate Abstract Syntax Tree

###### 5.l file(LEX file):

%{

#include"y.tab.h" #include<stdio.h> #include<string.h> int LineNo=1;

%}

identifier [a-zA-Z][\_a-zA-Z0-9]\* number [0-9]+|([0-9]\*\.[0-9]+)

%%

main\(\) return MAIN; if return IF;

else return ELSE; while return WHILE; int |

char |

float return TYPE;

{identifier} {strcpy(yylval.var,yytext); return VAR;}

{number} {strcpy(yylval.var,yytext); return NUM;}

\< |

\> |

\>= |

\<= |

== {strcpy(yylval.var,yytext); return RELOP;}

[ \t] ;

\n LineNo++;

. return yytext[0];

%%

###### 5.y(YACC FILE):

%{

#include<string.h> #include<stdio.h> struct quad

{

char op[5]; char arg1[10]; char arg2[10]; char result[10];

}QUAD[30];

struct stack

{

int items[100]; int top;

}stk;

int Index=0,tIndex=0,StNo,Ind,tInd; extern int LineNo;

%}

%union

{

char var[10];

}

%token <var> NUM VAR RELOP

%token MAIN IF ELSE WHILE TYPE

%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP

%left '-' '+'

%left '\*' '/'

%%

PROGRAM : MAIN BLOCK

;

BLOCK: '{' CODE '}'

;

CODE: BLOCK

| STATEMENT CODE

| STATEMENT

;

STATEMENT: DESCT ';'

| ASSIGNMENT ';'

| CONDST

| WHILEST

;

DESCT: TYPE VARLIST

;

VARLIST: VAR ',' VARLIST

| VAR

;

ASSIGNMENT: VAR '=' EXPR{

strcpy(QUAD[Index].op,"="); strcpy(QUAD[Index].arg1,$3); strcpy(QUAD[Index].arg2,""); strcpy(QUAD[Index].result,$1); strcpy($$,QUAD[Index++].result);

}

;

EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}

| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}

| EXPR '\*' EXPR {AddQuadruple("\*",$1,$3,$$);}

| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}

| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}

| '(' EXPR ')' {strcpy($$,$2);}

| VAR

| NUM

;

CONDST: IFST{

Ind=pop(); sprintf(QUAD[Ind].result,"%d",Index); Ind=pop(); sprintf(QUAD[Ind].result,"%d",Index);

}

| IFST ELSEST

;

IFST: IF '(' CONDITION ')' {

strcpy(QUAD[Index].op,"=="); strcpy(QUAD[Index].arg1,$3); strcpy(QUAD[Index].arg2,"FALSE"); strcpy(QUAD[Index].result,"-1"); push(Index);

Index++;

}

BLOCK { strcpy(QUAD[Index].op,"GOTO"); strcpy(QUAD[Index].arg1,""); strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,"-1"); push(Index);

Index++;

};

ELSEST: ELSE{

tInd=pop(); Ind=pop(); push(tInd);

sprintf(QUAD[Ind].result,"%d",Index);

} BLOCK{

Ind=pop(); sprintf(QUAD[Ind].result,"%d",Index);

};

CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);

StNo=Index-1;

}

| VAR

| NUM

;

WHILEST: WHILELOOP{

Ind=pop(); sprintf(QUAD[Ind].result,"%d",StNo); Ind=pop(); sprintf(QUAD[Ind].result,"%d",Index);

}

;

WHILELOOP: WHILE'('CONDITION ')' {

strcpy(QUAD[Index].op,"=="); strcpy(QUAD[Index].arg1,$3); strcpy(QUAD[Index].arg2,"FALSE"); strcpy(QUAD[Index].result,"-1"); push(Index);

Index++;

} BLOCK {

strcpy(QUAD[Index].op,"GOTO"); strcpy(QUAD[Index].arg1,"");

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,"-1"); push(Index);

Index++;

}

;

%%

extern FILE \*yyin;

int main(int argc,char \*argv[])

{

FILE \*fp; int i; if(argc>1)

{

fp=fopen(argv[1],"r"); if(!fp)

{

printf("\n File not found"); exit(0);

}

yyin=fp;

}

yyparse();

printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator \tArg1 \tArg2 \tResult" "\n\t\t---------------

");

for(i=0;i<Index;i++)

{

printf("\n\t\t %d\t %s\t %s\t %s\t%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);

}

printf("\n\t\t ");

printf("\n\n"); return 0; } void push(int data)

{ stk.top++; if(stk.top==100)

{

printf("\n Stack overflow\n"); exit(0);

}

stk.items[stk.top]=data;

}

int pop()

{

int data; if(stk.top==-1)

{

printf("\n Stack underflow\n"); exit(0);

}

data=stk.items[stk.top--]; return data;

}

void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])

{

strcpy(QUAD[Index].op,op); strcpy(QUAD[Index].arg1,arg1); strcpy(QUAD[Index].arg2,arg2); sprintf(QUAD[Index].result,"t%d",tIndex++); strcpy(result,QUAD[Index++].result);

}

yyerror()

{

printf("\n Error on line no:%d",LineNo);

}

###### INPUT:

main()

{

int a,b,c; if(a<b)

{

a=a+b;

}

while(a<b)

{

a=a+b;

}

if(a<=b)

{

c=a-b;

}

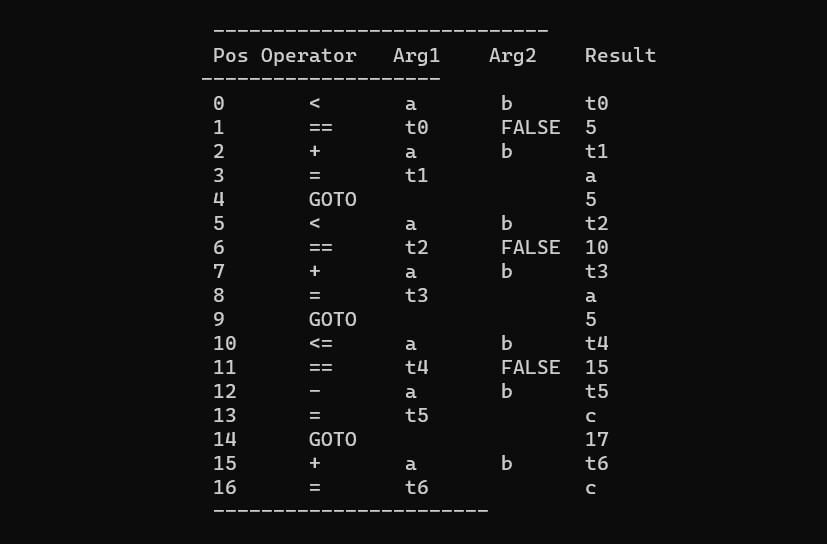
else

{

c=a+b;

}

}



# Week 6:

### Convert the BNF rules into YACC form and write code to generate Abstract Syntax Tree

###### 5.l file(LEX file):

%{

#include"y.tab.h" #include<stdio.h> #include<string.h> int LineNo=1;

%}

identifier [a-zA-Z][\_a-zA-Z0-9]\* number [0-9]+|([0-9]\*\.[0-9]+)

%%

main\(\) return MAIN; if return IF;

else return ELSE; while return WHILE; int |

char |

float return TYPE;

{identifier} {strcpy(yylval.var,yytext); return VAR;}

{number} {strcpy(yylval.var,yytext); return NUM;}

\< |

\> |

\>= |

\<= |

== {strcpy(yylval.var,yytext); return RELOP;}

[ \t] ;

\n LineNo++;

. return yytext[0];

%%

###### 5.y(YACC FILE):

%{

#include<string.h> #include<stdio.h> struct quad

{

char op[5]; char arg1[10]; char arg2[10]; char result[10];

}QUAD[30];

struct stack

{

int items[100]; int top;

}stk;

int Index=0,tIndex=0,StNo,Ind,tInd; extern int LineNo;

%}

%union

{

char var[10];

}

%token <var> NUM VAR RELOP

%token MAIN IF ELSE WHILE TYPE

%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP

%left '-' '+'

%left '\*' '/'

%%

PROGRAM : MAIN BLOCK

;

BLOCK: '{' CODE '}'

;

CODE: BLOCK

| STATEMENT CODE

| STATEMENT

;

STATEMENT: DESCT ';'

| ASSIGNMENT ';'

| CONDST

| WHILEST

;

DESCT: TYPE VARLIST

;

VARLIST: VAR ',' VARLIST

| VAR

;

ASSIGNMENT: VAR '=' EXPR{

strcpy(QUAD[Index].op,"="); strcpy(QUAD[Index].arg1,$3); strcpy(QUAD[Index].arg2,""); strcpy(QUAD[Index].result,$1); strcpy($$,QUAD[Index++].result);

}

;

EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}

| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}

| EXPR '\*' EXPR {AddQuadruple("\*",$1,$3,$$);}

| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}

| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}

| '(' EXPR ')' {strcpy($$,$2);}

| VAR

| NUM

;

CONDST: IFST{

Ind=pop(); sprintf(QUAD[Ind].result,"%d",Index); Ind=pop(); sprintf(QUAD[Ind].result,"%d",Index);

}

| IFST ELSEST

;

IFST: IF '(' CONDITION ')' {

strcpy(QUAD[Index].op,"=="); strcpy(QUAD[Index].arg1,$3); strcpy(QUAD[Index].arg2,"FALSE"); strcpy(QUAD[Index].result,"-1"); push(Index);

Index++;

}

BLOCK { strcpy(QUAD[Index].op,"GOTO"); strcpy(QUAD[Index].arg1,""); strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,"-1"); push(Index);

Index++;

};

ELSEST: ELSE{

tInd=pop(); Ind=pop(); push(tInd);

sprintf(QUAD[Ind].result,"%d",Index);

} BLOCK{

Ind=pop(); sprintf(QUAD[Ind].result,"%d",Index);

};

CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);

StNo=Index-1;

}

| VAR

| NUM

;

WHILEST: WHILELOOP{

Ind=pop(); sprintf(QUAD[Ind].result,"%d",StNo); Ind=pop(); sprintf(QUAD[Ind].result,"%d",Index);

}

;

WHILELOOP: WHILE'('CONDITION ')' {

strcpy(QUAD[Index].op,"=="); strcpy(QUAD[Index].arg1,$3); strcpy(QUAD[Index].arg2,"FALSE"); strcpy(QUAD[Index].result,"-1"); push(Index);

Index++;

} BLOCK {

strcpy(QUAD[Index].op,"GOTO"); strcpy(QUAD[Index].arg1,"");

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,"-1"); push(Index);

Index++;

}

;

%%

extern FILE \*yyin;

int main(int argc,char \*argv[])

{

FILE \*fp; int i; if(argc>1)

{

fp=fopen(argv[1],"r"); if(!fp)

{

printf("\n File not found"); exit(0);

}

yyin=fp;

}

yyparse();

printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator \tArg1 \tArg2 \tResult" "\n\t\t---------------

");

for(i=0;i<Index;i++)

{

printf("\n\t\t %d\t %s\t %s\t %s\t%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);

}

printf("\n\t\t ");

printf("\n\n"); return 0; } void push(int data)

{ stk.top++; if(stk.top==100)

{

printf("\n Stack overflow\n"); exit(0);

}

stk.items[stk.top]=data;

}

int pop()

{

int data; if(stk.top==-1)

{

printf("\n Stack underflow\n"); exit(0);

}

data=stk.items[stk.top--]; return data;

}

void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])

{

strcpy(QUAD[Index].op,op); strcpy(QUAD[Index].arg1,arg1); strcpy(QUAD[Index].arg2,arg2); sprintf(QUAD[Index].result,"t%d",tIndex++); strcpy(result,QUAD[Index++].result);

}

yyerror()

{

printf("\n Error on line no:%d",LineNo);

}

###### INPUT:

main()

{

int a,b,c; if(a<b)

{

a=a+b;

}

while(a<b)

{

a=a+b;

}

if(a<=b)

{

c=a-b;

}

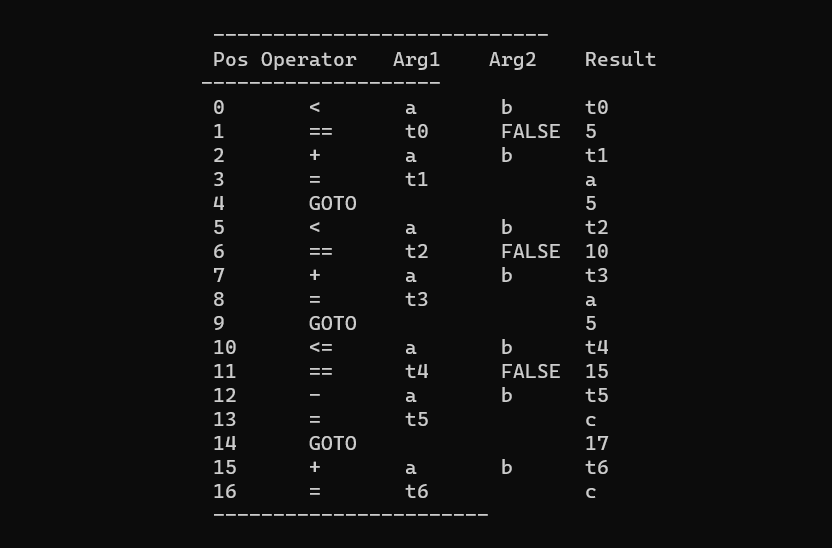
else

{

c=a+b;

}

}



**WEEK 7**

C code:

#include<stdio.h> #include<conio.h> #include<string.h>

char op[2],arg1[5],arg2[5],result[5]; void main()

{

FILE \*fp1,\*fp2; fp1=fopen("input.txt","r");

fp2=fopen("output.txt","w"); while(!feof(fp1)){

fscanf(fp1,"%s%s%s%s",op,arg1,arg2,result); if(strcmp(op,"+")==0){

fprintf(fp2,"\nMOV R0,%s",arg1); fprintf(fp2,"\nADD R0,%s",arg2); fprintf(fp2,"\nMOV %s,R0",result);

}

if(strcmp(op,"\*")==0){ fprintf(fp2,"\nMOV R0,%s",arg1); fprintf(fp2,"\nMUL R0,%s",arg2); fprintf(fp2,"\nMOV %s,R0",result);

}

if(strcmp(op,"-")==0){ fprintf(fp2,"\nMOV R0,%s",arg1); fprintf(fp2,"\nSUB R0,%s",arg2); fprintf(fp2,"\nMOV %s,R0",result);

}

if(strcmp(op,"/")==0){ fprintf(fp2,"\nMOV R0,%s",arg1);

fprintf(fp2,"\nDIV R0,%s",arg2); fprintf(fp2,"\nMOV %s,R0",result);

}

if(strcmp(op,"=")==0){ fprintf(fp2,"\nMOV R0,%s",arg1); fprintf(fp2,"\nMOV %s,R0",result);

}

}

fclose(fp1); fclose(fp2); getch();

}

