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.

(a) 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");
   }
   return 0;
}
```

```
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_a.c
Enter the string:
abc
The given string is not constant
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_a.c
Enter the string:
1234
The given string is constant
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$
```

(b)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{
                printf("Given statement is not a comment\n");
            }
        }
        else{
            printf("Given statement is not a comment\n");
        }
    }
    else{
        printf("Given statement is not a comment\n");
    }
    return 0;
}
```

```
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_b.c
Enter the string:
//comment
Given statement is a comment
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_b.c
Enter the string:
/hfuhsf
Given statement is not a comment
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_b.c
Enter the string:
/*comment*/
Given statement is a comment
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_b.c
Enter the string:
/*idhffhg
Given statement is not a comment
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$
```

(c)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;
}
```

```
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_c.c
Enter the string:
abcd
Given string is valid identifier
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_c.c
Enter the string:
_name
Given string is valid identifier
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_c.c
Enter the string:
3ghg
Given string is not valid!
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_c.c
Enter the string:
abc1
Given string is valid identifier
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$
```

(d)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;
}
```

```
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_d.c
for (i=0;i<n;i++) if (i%2 == 0) {} else {}
for is a keyword
if is a keyword
else is a keyword
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_d.c
char s=0;
char is a keyword
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$
```

(e)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
                printf("Greater than\n");
            break;}
        case '<':{
            if (s[1] == '=')
                printf("Less than Equal to\n");
            else
                printf("Less than\n");
            break;}
        case '=':{
            if (s[1] == '=')
                printf("Comparing operator\n");
            else
                printf("Assignment operator\n");
            break;}
        case '|':{
            if (s[1] == '|')
                printf("Logical OR\n");
            else
                printf("Bitwise OR\n");
            break;}
        case '&':{
            if (s[1] == '&')
```

```
                        printf("Logical AND\n");
                else
                        printf("Bitwises AND\n");
                break;}
        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;
}
```

```
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_e.c
Enter the operator:
-
Subtraction operator
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_e.c
Enter the operator:
<=
Less than Equal to
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$ ./a.out week1
_e.c
Enter the operator:
|
Bitwise OR
vnrvjiet@vnrvjiet-HP-ProDesk-400-G7-Microtower-PC:~/20071A6955_CD$
```

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:

```
ubuntu-nslab@ubuntunslab:~$ ./a.out f1.c

/ is a Binary Operator
/ is a Binary Operator
var is a Identifier
. is a Binary Operator
c is a Identifier#
include is a Keyword
 < is an integer constant
stdio is a Identifier
. is a Binary Operator
h is a Identifier>#
include is a Keyword
 < is an integer constant
conio is a Identifier
. is a Binary Operator
h is a Identifier>
void is a Identifier
main is a Identifier
( is a Special Symbol
) is a Special Symbol
{ is a Special Symbol
```

```
int is a Keyword
a is a Identifier
, is a Special Symbol
b is a Identifier
, is a Special Symbol
c is a Identifier
; is a Special Symbol
a is a Identifier
= is a Assignment operator
 1 is an integer constant
; is a Special Symbol
b is a Identifier
= is a Assignment operator
 2 is an integer constant
; is a Special Symbol
c is a Identifier
= is a Assignment operator
a is a Identifier
+ is a Binary Operator
b is a Identifier
; is a Special Symbol
printf is a Identifier
( is a Special Symbol
 "Sum:%d" is a literal
, is a Special Symbol
c is a Identifier
) is a Special Symbol
; is a Special Symbol
} is a Special Symbolubuntu-nslab@ubuntunslab:~$
```

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:

```
input
The following grammar is used for Parsing Table:
S->A
A->Bb
A->Cd
B->aB
B->@
C->Cc
C->@

Predictive parsing table:

-------------------------------------------------------
          a         b         c         d         $
-------------------------------------------------------
S         S->A      S->A      S->A      S->A
-------------------------------------------------------
A         A->Bb     A->Bb     A->Cd     A->Cd
-------------------------------------------------------
B         B->aB     B->@      B->@                B->@
-------------------------------------------------------
C                             C->@      C->@      C->@
-------------------------------------------------------


...Program finished with exit code 0
Press ENTER to exit console.
```

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:

```
syntax errorubuntu-nslab@ubuntunslab:~$ lex week4.l
ubuntu-nslab@ubuntunslab:~$ yacc -d week4.y
ubuntu-nslab@ubuntunslab:~$ cc lex.yy.c y.tab.c -ll -lm
y.tab.c: In function ‘yyparse’:
y.tab.c:1225:16: warning: implicit declaration of function ‘yylex’ [-Wimplicit-
function-declaration]
 1225 |       yychar = yylex ();
      |                ^~~~~
y.tab.c:1378:7: warning: implicit declaration of function ‘yyerror’; did you me
an ‘yyerrok’? [-Wimplicit-function-declaration]
 1378 |       yyerror (YY_("syntax error"));
      |       ^~~~~~~
      |       yyerrok
week4.y: At top level:
week4.y:34:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
   34 | yyerror(char *s)
      | ^~~~~~~
ubuntu-nslab@ubuntunslab:~$ ./a.out
2+3
5
```

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;
}
}
```

```
sahithi@LAPTOP-2PTH9I8T:~$ gcc lex.yy.c y.tab.c -ll -lm -w
sahithi@LAPTOP-2PTH9I8T:~$ ./a.out 5.c


                    ----------------------------
                    Pos Operator   Arg1    Arg2    Result
                   --------------------
                    0       <       a       b       t0
                    1       ==      t0      FALSE   5
                    2       +       a       b       t1
                    3       =       t1              a
                    4       GOTO                    5
                    5       <       a       b       t2
                    6       ==      t2      FALSE   10
                    7       +       a       b       t3
                    8       =       t3              a
                    9       GOTO                    5
                    10      <=      a       b       t4
                    11      ==      t4      FALSE   15
                    12      -       a       b       t5
                    13      =       t5              c
                    14      GOTO                    17
                    15      +       a       b       t6
                    16      =       t6              c
                    ------------------------
```

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;
}
}
```

```
sahithi@LAPTOP-2PTH9I8T:~$ gcc lex.yy.c y.tab.c -ll -lm -w
sahithi@LAPTOP-2PTH9I8T:~$ ./a.out 5.c


                    ----------------------------
                    Pos Operator   Arg1    Arg2    Result
                   --------------------
                    0       <       a       b       t0
                    1       ==      t0      FALSE   5
                    2       +       a       b       t1
                    3       =       t1              a
                    4       GOTO                    5
                    5       <       a       b       t2
                    6       ==      t2      FALSE   10
                    7       +       a       b       t3
                    8       =       t3              a
                    9       GOTO                    5
                    10      <=      a       b       t4
                    11      ==      t4      FALSE   15
                    12      -       a       b       t5
                    13      =       t5              c
                    14      GOTO                    17
                    15      +       a       b       t6
                    16      =       t6              c
                    ------------------------
```

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();
}
```

input.txt

```
+ a b t1
* c d t2
- t1 t2 t
= t ? x
```

output.txt

```
MOV R0,a
ADD R0,b
MOV t1,R0
MOV R0,c
MUL R0,d
MOV t2,R0
MOV R0,t1
SUB R0,t2
MOV t,R0
MOV R0,t
MOV x,R0
```