1. **Write a C Program for dividing the input program into lexemes.**

File Name:Exp1.c

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
void main(){
 FILE *fi, *fo, *fop, *fk;
 int flag=0, i=1;
 char c,a[15],ch[15],file[20];
 char name[20];
 char rollno[15];
 printf("Enter name: ");
 scanf("%s",name);
 printf("Enter rollno: ");
 scanf("%s",rollno);
 printf("\n Enter the FileName: ");
 scanf("%s",&file);
 fi=fopen(file,"r");
 fo=fopen("lexemes.c","w");
 fop=fopen("o.c","r");
 fk=fopen("k.c","r");
 c=getc(fi);
 while(!feof(fi)){
    if(isalpha(c)||isdigit(c)||(c=='['||c==']'||c=='.'==1))
                fputc(c,fo);
    else{
          if(c=='\n')
                fprintf(fo,"\t$\t");
```

```
        else
            fprintf(fo,"\t%c\t",c);
    }
    c=getc(fi);
}
fclose(fi);
fclose(fo);
fi=fopen("lexemes.c","r");
printf("\nName of the student: %s",name);
printf("\nRollNo of the student: %s",rollno);
printf("\nLexical Analysis");
fscanf(fi,"%s",a);
printf("\nLine:%d\n",i++);
while(!feof(fi)){
        if(strcmp(a,"$")==0){
            printf("\nLine:%d\n",i++);
            fscanf(fi,"%s",a);
        }
        fscanf(fop,"%s",ch);
        while(!feof(fop)){
           if(strcmp(ch,a)==0){
                fscanf(fop,"%s",ch);
                printf("\t\t%s\t:\t%s\n",a,ch);
                flag=1;
           }
           fscanf(fop,"%s",ch);
        }
        rewind(fop);
        fscanf(fk,"%s",ch);
```

```
        while(!feof(fk)){
            if(strcmp(ch,a)==0){
                fscanf(fk,"%k",ch);
                printf("\t\t%s\t:\tKeywords\n",a); //
                flag=1;
            }
            fscanf(fk,"%s",ch);
        }
        rewind(fk);
        if(flag==0){
            if(isdigit(a[0]))
               printf("\t\t%s\t:\tConstant\n",a);
            else
                printf("\t\t%s\t:\tidentifier\n",a);
        }
        flag=0;
        fscanf(fi,"%s",a);
   }
getch();
}
```

File Name:i.c

```
#include<stdio.h>
#include<conio.h>
void main(){
   int a=10,b,c;
   a=b*c;
   getch();
}
```

File Name:o.c

(operator

) closeoperator

{ open brace

} close brace

< lessthan operator

> greaterthan operator

" doublequotes operator

' singlequotes operator

: colon

; semicon

prepocesser directive

= equal to operator

== assignment operator

% percentage symbol

^ bitwise operator

& referene symbol

+ addition operator

- subtraction operator

* multiplication operator

\ backslah

/ slash

, comma

stdio.h headerfile

conio.h headerfile

! not equal to operator

File Name:k.c

auto

break

case

char

const

continue

default

do

double

else

enum

extern

float

for

goto

if

int

long

register

return

short

signed

sizeof

static

struct

switch

typedef

union

unsigned

void

volatile

while

2. **Write a C Program to Simulate FIRST and FOLLOW of grammar.**

File Name:exp2.c

```
#include<stdio.h>
#include<math.h>
#include<string.h>
#include<ctype.h>
#include<stdlib.h>
int n,m=0,i=0,j=0;
char a[10][10],f[10];
void follow(char c);
void refirst(char c);
void first(char c);
int main()
{
int i,z;
char c,ch;
char name[20];
char rollno[15];
printf("enter the name:");
scanf("%s",name);
printf("enter roll no:");
scanf("%s",rollno);
printf("\n Enter the no of prooductions:\n");
scanf("%d",&n);
printf("Enter the productions:\n");
```

```
for(i=0;i<n;i++)
scanf("%s%c",a[i],&ch);
do
{
m=0;
printf("Enter the elemets whose fisrt & follow is to be found:");
scanf("%c",&c);
first(c);
printf("First(%c)={",c);
for(i=0;i<m;i++)
printf("%c",f[i]);
printf("}\n");
strcpy(f," ");
//flushall();
m=0;
follow(c);
printf("Follow(%c)={",c);
for(i=0;i<m;i++)
printf("%c",f[i]);
printf("}\n");
printf("Continue(0/1)?");
scanf("%d%c",&z,&ch);
}
while(z==1);
return(0);
}
void refirst(char c)
{
int j;
```

```
if(!(isupper(c)))
f[m++]=c;
for(j=0;j<n;j++)
{
if(a[j][0]==c)
{
if(a[j][2]=='$')
{
follow(a[j][0]);
}
else if(islower(a[j][2]))
f[m++]=a[j][2];
else
refirst(a[j][2]);
}
}
}
void first(char c)
{
int j;
if(!(isupper(c)))
f[m++]=c;
for(j=0;j<n;j++)
{
if(a[j][0]==c)
{
if(a[j][2]=='$')
{
f[m++]='$';
```

```
}
else if(islower(a[j][2]))
f[m++]=a[j][2];
else
first(a[j][2]);
}
}
}
void follow(char c)
{ if(a[0][0]==c)
f[m++]='$';
for(i=0;i<n;i++)
{
for(j=2;j<strlen(a[i]);j++)
{
if(a[i][j]==c)
{
if(a[i][j+1]!='\0')
refirst(a[i][j+1]);
if(a[i][j+1]=='\0' && c!=a[i][0])
follow(a[i][0]);
}
}
}
}
```

3. **Write a C program for implementing the operator precedence.**

File Name:exp3.c

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

```
void main()
{
        char W[20],STACK[20],TERMINALS[20],PRECEDENCE_TABLE[20][20][1];
    char name[20];
    char rollno[15];
        int TERMINALS_N,i,j,k,IP = 0,TOP = 0,FLAG = 1,COLUMN,ROW;
        for(i=0;i<20;i++)
        {
                STACK[i]=(int)NULL;
                W[i]=(int)NULL;
                for(j=0;j<20;j++)
                {
                        PRECEDENCE_TABLE[i][j][1]=(int)NULL;
                }
        }
        printf("INPUT STRING WITH DOLLAR ($) SIGN  : ");
        gets(W);
        printf("enter the name of the student:");
        scanf("%s",name);
        printf("\nenter the roll no of the student:");
        scanf("%s",&rollno);
        printf("\nthe name of the student is :%s",name);
        printf("\nthe rollno of the student is :%s",rollno);
        printf("\nNO. OF TERMINALS  : ");
        scanf("%d",&TERMINALS_N);
        printf("\nENTER TERMINALS (SERIALLY WITHOUT SPACES) :\n");
        scanf("%s",TERMINALS);
        printf("\nENTER OPERATOR PRECEDENCE TABLE  :\n\n");
        for(i=0;i<TERMINALS_N;i++)
```

```
	{
		for(j=0;j<TERMINALS_N;j++)
		{
			printf("ENTER PRECEDENCE FOR %c AND %c :
",TERMINALS[i],TERMINALS[j]);
			_flushall();
			scanf("%c",&PRECEDENCE_TABLE[i][j]);
		}
	}
	printf("\n\n----- OPERATOR PRECEDENCE TABLE -----\n\n");
	for(i=0;i<TERMINALS_N;i++)
	{
		printf("\t%c",TERMINALS[i]);
	}
	printf("\n");
	for(i=0;i<TERMINALS_N;i++)
	{
		printf("\n%c",TERMINALS[i]);
		for(j=0;j<TERMINALS_N;j++)
		{
			printf("\t%c",PRECEDENCE_TABLE[i][j][0]);
		}
	}
	STACK[TOP] = '$';
	printf("\n");
	printf("\n\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");
	printf("\n%s\t\t\t%s\t\t\t",STACK,W);
	while(IP <= strlen(W))
	{
		for(k=0;k<TERMINALS_N;k++)
```

```
		{
			if(STACK[TOP] == TERMINALS[k])
			{
				COLUMN = k;
			}
			if(W[IP] == TERMINALS[k])
			{
				ROW = k;
			}
		}
		if((STACK[TOP] == '$') && (W[IP] == '$'))		//SUCCESSFUL PARSING
		{
			printf("\n\n\n\t----- INPUT STRING SUCCESSFULLY PARSED -----\n");
			break;
		}
		else if((PRECEDENCE_TABLE[COLUMN][ROW][0] == '<') ||
(PRECEDENCE_TABLE[COLUMN][ROW][0] == '='))
		//PUSHING IF STACK ELEMENT IS LESS THAN OR EQUAL TO INPUT STRING
ELEMENT
		{
			STACK[++TOP] = PRECEDENCE_TABLE[COLUMN][ROW][0];
	//DEFINITELY NEEDED
			STACK[++TOP] = W[IP];
			printf("SHIFT OPERATION ON  %c",W[IP]);
			IP++;
		}
		else		//POPPING IF STACK ELEMENT IS GREATER THAN INPUT
STRING ELEMENT
		{
			if(PRECEDENCE_TABLE[COLUMN][ROW][0] == '>')
			{
```

```
                        while(STACK[TOP] != '<')
                        {
                                --TOP;
                        }
                        TOP--;                          //MAY NOT BE NEEDED
                        printf("REDUCE OPERATION");
                }
                else
                {
                        printf("\n\n----- UNABLE TO PARSE STRING -----");
                        break;
                }
        }
        printf("\n");
        for(i=0;i<=TOP;i++)
        {
                printf("%c",STACK[i]);
        }
        printf("\t\t\t");
        for(k=IP;k<strlen(W);k++)
        {
                printf("%c",W[k]);
        }
        printf("\t\t\t");
    }
}
```

4. **Design a lexical analyzer for the given language. The lexical analyzer should ignore redundant spaces, tabs and new lines, comments etc.**

File name :exp4.c

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
// Token types
typedef enum
{
IDENTIFIER,
NUMBER,
OPERATOR,
COMMENT,
WHITESPACE,
NEWLINE,
EOF_TOKEN
}
TokenType;
// Token structure
typedef struct
{
TokenType type;
char value[100];
} Token;
// Lexical analyzer function
Token getNextToken(char *input, int *pos)
{
Token token;
char c = input[*pos];
token.value[0] = '\0';
// Skip whitespace
while (isspace(c))
```

```
{
(*pos)++;
c = input[*pos];
}
// Skip comments
if (c == '/')
{
if (input[*pos + 1] == '/')
{
while (c != '\n')
{
(*pos)++;
c = input[*pos];
}
return
getNextToken(input, pos);
}
else if (input[*pos + 1] == '*')
{
 while (!(input[*pos] == '*' && input[*pos + 1] == '/'))
 {
 (*pos)++;
 }
 (*pos) += 2;
 return
getNextToken(input, pos);
 }
 }
// End of file
```

```
if (c == '\0')
{
token.type = EOF_TOKEN;
return token;
}
 // Identifier or keyword
if (isalpha(c) || c == '_')
{
int i = 0;
while (isalnum(c) || c == '_')
{
 token.value[i++] = c;
(*pos)++;
c = input[*pos];
}
token.value[i] = '\0';
token.type = IDENTIFIER;
return token;
}
// Number
if (isdigit(c) || c == '.')
{
int i = 0;
while (isdigit(c) || c == '.')
{
token.value[i++] = c;
(*pos)++;
c = input[*pos];
}
```

```
token.value[i] = '\0';
token.type = NUMBER;
return token;
}
// Operator if (c == '+' || c == '-' || c == '*' || c == '/')
{
token.value[0] = c;
token.value[1] = '\0';
token.type = OPERATOR;
(*pos)++;
return token;
}
// Newline
if (c == '\n')
{
token.value[0] = '\n';
token.value[1] = '\0';
token.type = NEWLINE;
(*pos)++;
return token;
}
// Other characters
token.value[0] = c;
token.value[1] = '\0';
(*pos)++;
return token;
}
int main()
{
```

```
char name[20];
char rollno[20];
printf("\n Enter the Name:");
scanf("%s",name);
printf("\n Enter the Roll No:");
scanf("%s",rollno);
char input[] = "int main() {\n int a=18;\n if (a>18){ printf(a);\n}\n else: \n { printf(a+5);\n };\n}";
int pos = 0;
Token token;
printf("---------------------\n");
printf("\n OUTPUT:\n");
printf("---------------------\n");
do
{
token = getNextToken(input, &pos);
switch (token.type)
{
case IDENTIFIER:
printf("Identifier: %s\n", token.value);
break;
case NUMBER:
printf("Number: %s\n", token.value);
break;
case OPERATOR:
printf("Operator: %s\n", token.value);
break;
case COMMENT:
printf("Comment: %s\n", token.value);
break;
```

```
case WHITESPACE:
// Ignore whitespace tokens
break;
case NEWLINE:
printf("Newline\n");
break;
case EOF_TOKEN:
printf("End of file\n");
break;
default:
printf("Unknown token\n");
break;
}
}
while (token.type != EOF_TOKEN);
return 0;
}
```

5. Write a C Program to Implement a Recursive Descent Parser.

File Name:exp5.c

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
char input[100];
char name[20];
char rollno[20];
int i;
int S();
int A();
int main(void)
```

```
{
printf("\nRecursive Descent Parsing for the following grammar\n\n");
printf("\nS->cAd'\nA->ab/a\n\n");
printf("\nEnter the String to be Checked:");
gets(input);

printf("\n Enter the Name:");
scanf("%s",name);
printf("\n Enter the Roll No:");
scanf("%s",rollno);
if(S())
{
if(input[i+1]=='\0')
printf("\nString is ACCEPTED");
else
printf("\nString is NOT ACCEPTED");
}
else
printf("\nString is NOT ACCEPTED");

}
int S()
{
if(input[i]=='c')
{
 i++;
 if(A())
 if(input[i]=='d')
 i++;
```

```
 return 1;
}
 else
 return 0;
}
int A()
{
if(input[i]=='a')
{
 i++;
 if(input[i]=='b')
 i++;
}
else if(input[i]=='a')
i++;
return 1;
}
```

6. **Write a C program to construct LL (1) parser.**

FileName:exp6.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char prod[3][10]={"A->aBa","B->bB","B->&"};
char first[3][10]={"a","b","&"};
char follow[3][10]={"$","a","a"};
char table[3][4][10];
char name[20];
char rollno[20];
char input[10];
```

```
int top=-1;
char stack[25];
char curprod[20];
void push(char item)
{
stack[++top]=item;
}
void pop()
{
top=top-1;
}
void display()
{
int i;
for(i=top;i>=0;i--)
printf("%c",stack[i]);
}
int numeric(char c)
{
switch(c)
{
case'A': return 1;
case'B': return 2;
case'a': return 1;
case'b': return 2;
case'$': return 3;
default: return 3;
}
return 1;
```

```
}
int main()
{
printf("\nEnter the Name of the Student:");
scanf("%s",name);
printf("Enter Rollno of the Student:");
scanf("%s",rollno);
printf("\nName of the Student  is :%s",name);
printf("\nRollno of the Student is :%s",rollno);
char c;
int i,j,k,n;
for(i=0;i<3;i++)
{
for(j=0;j<4;j++)
{
strcpy(table[i][j],"EMPTY");
}
}
printf("\n\nTHE GIVEN GRAMMAR IS:\n");
for(i=0;i<3;i++)
printf("%s\n",prod[i]);
printf("\nFIRST AND FOLLOW OF THE NON TERMINAL SYMBOLS IN THE GIVEN GRAMMAR ARE:");
printf("\nFIRST={%s,%s,%s}",first[0],first[1],first[2]);
printf("\nFOLLOW={%s,%s}\n",follow[0],follow[1]);
printf("\nLL(1) PARSE TABLE FOR THE GIVEN GRAMMAR IS:\n");
strcpy(table[0][0],"");
strcpy(table[0][1],"a");
strcpy(table[0][2],"b");
strcpy(table[0][3],"$");
```

```
strcpy(table[1][0],"A");
strcpy(table[2][0],"B");
for(i=0;i<3;i++)
{
if(first[i][0]!='&')
strcpy(table[numeric(prod[i][0])][numeric(first[i][0])],prod[i]);
else
strcpy(table[numeric(prod[i][0])][numeric(follow[i][0])],prod[i]);
}
printf("\n---------- \n");
for(i=0;i<3;i++)
{
for(j=0;j<4;j++)
{
printf("%-30s",table[i][j]);
if(j==3)
printf("\n----------\n");
}
}
printf("\n Enter the Input String Terminated with $ to Parse:");
scanf("%s",input);
for(i=0;input[i]!='\0';i++)
{
if((input[i]!='a')&&(input[i]!='b')&&(input[i]!='$'))
{
printf("\nINVALID STRING ENTERED");
exit(0);
}
}
```

```
push('$');
push('A');
i=0;
printf("\n\n");
printf("STACK\t\t INPUT\t\t\t Action");
printf("\n-----------------------------------------------------------\n");
while(stack[top]!='$')
{
display();
printf("\t\t%s\t",(input+i));
if(stack[top]==input[i])
{
printf("\tMatched %c\n",input[i]);
pop();
i++;
}
else
{
if(stack[top]>=65&&stack[top]<92)
{
strcpy(curprod,table[numeric(stack[top])][numeric(input[i])]);
if(!(strcmp(curprod,"EMPTY")))
{
printf("\n UNABLE TO PARSE THE STRING \n");
exit(0);
}
else
{
printf("\tApply Production %s\n",curprod);
```

```
if (curprod[3]=='&')
pop();
else
{
pop();
n=strlen(curprod);
for(j=n-1;j>=3;j--)
push(curprod[j]);
}
}
}
}
}
display();
printf("\t\t%s\t",(input+i));
printf("\n---------------------------------------------\n");
if(stack[top]=='$'&&input[i]=='$')
{
printf("\n STRING IS SUCCESSFULLY PARSED\n");
}
else if(stack[top]=='$' || input[i]=='$')
{
printf("\nUNABLE TO PARSE THE STRING\n");
}
}
```

7. Write a C Program to Implement a Predictive Parser.

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

```
char prod[3][10]={"S->aBA","A->bB","B->c"};
char first[3][10]={"a","b","c"};
char follow[3][10]={"$","$","b"};
char table[4][5][10];
char name[20];
char rollno[20];
char input[10];
int top=-1;
char stack[25];
char curprod[20];
void push(char item)
{
stack[++top]=item;
}
void pop()
{
top=top-1;
}
void display()
{
int i;
for(i=top;i>=0;i--)
printf("%c",stack[i]);
}
int numeric(char c)
{
switch(c)
{
case'S': return 1;
```

```
case'A': return 2;
case'B': return 3;
case'a': return 1;
case'b': return 2;
case'c': return 3;
case'$': return 4;
default: return 4;
}
return 1;
}
int main()
{
printf("\nEnter the Name of the Student:");
scanf("%s",name);
printf("Enter Rollno of the Student:");
scanf("%s",rollno);
printf("\nName of the Student  is :%s",name);
printf("\nRollno of the Student is :%s",rollno);
char c;
int i,j,k,n;
for(i=0;i<4;i++)
{
for(j=0;j<5;j++)
{
strcpy(table[i][j],"EMPTY");
}
}
printf("\n\nTHE GIVEN GRAMMAR IS:\n");
for(i=0;i<4;i++)
```

```
printf("%s\n",prod[i]);
printf("\nFIRST AND FOLLOW OF THE NON TERMINAL SYMBOLS IN THE GIVEN GRAMMAR ARE:");
printf("\nFIRST={%s,%s,%s}",first[0],first[1],first[2]);
printf("\nFOLLOW={%s,%s}\n",follow[0],follow[2]);
printf("\nPREDICTIVE PARSE TABLE FOR THE GIVEN GRAMMAR IS:\n");
strcpy(table[0][0],"");
strcpy(table[0][1],"a");
strcpy(table[0][2],"b");
strcpy(table[0][3],"c");
strcpy(table[0][4],"$");
strcpy(table[1][0],"S");
strcpy(table[2][0],"A");
strcpy(table[3][0],"B");
for(i=0;i<4;i++)
{
if(first[i][0]!='&')
strcpy(table[numeric(prod[i][0])][numeric(first[i][0])],prod[i]);
else
strcpy(table[numeric(prod[i][0])][numeric(follow[i][0])],prod[i]);
}
printf("\n---------- \n");
for(i=0;i<4;i++)
{
for(j=0;j<5;j++)
{
printf("%-30s",table[i][j]);
if(j==4)
printf("\n----------\n");
}
```

```
}
printf("\n Enter the Input String Terminated with $ to Parse:");
scanf("%s",input);
for(i=0;input[i]!='\0';i++)
{
if((input[i]!='a')&&(input[i]!='b')&&(input[i]!='c')&&(input[i]!='$'))
{
printf("\nINVALID STRING ENTERED");
exit(0);
}
}
push('$');
push('S');
i=0;
printf("\n\n");
printf("STACK\t\t INPUT\t\t\t Action");
printf("\n---------------------------------------------------------\n");
while(stack[top]!='$')
{
display();
printf("\t\t%s\t",(input+i));
if(stack[top]==input[i])
{
printf("\tMatched %c\n",input[i]);
pop();
i++;
}
else
{
```

```
if(stack[top]>=65&&stack[top]<92)
{
strcpy(curprod,table[numeric(stack[top])][numeric(input[i])]);
if(!(strcmp(curprod,"EMPTY")))
{
printf("\n UNABLE TO PARSE THE STRING \n");
exit(0);
}
else
{
printf("\tApply Production %s\n",curprod);
if (curprod[4]=='$')
pop();
else
{
pop();
n=strlen(curprod);
for(j=n-1;j>=4;j--)
push(curprod[j]);
}
}
}
}
}
display();
printf("\t\t%s\t",(input+i));
printf("\n---------------------------------------------\n");
if(stack[top]=='$'&&input[i]=='$')
{
```

```
printf("\n STRING IS SUCCESSFULLY PARSED\n");
}
else if(stack[top]=='$' || input[i]=='$')
{
printf("\nUNABLE TO PARSE THE STRING\n");
}
}
```

8. **Write a C Program to construct shift- reduce parser.**

```
#include <stdio.h>
#include <string.h>
struct ProductionRule
{
   char left[10];
   char right[10];
};
int main()
{
int flag=0;
   char input[20], stack[50], temp[50], ch[2], *token1, *token2,
*substring;
   int i, j, stack_length, substring_length, stack_top, rule_count = 0;
   struct ProductionRule rules[10];
   stack[0] = '\0';
   printf("\nEnter the Number of Productions: ");
   scanf("%d", &rule_count);
   printf("\nEnter the Productions: \n");
   for (i = 0; i < rule_count; i++)
   {
      scanf("%s", temp);
      token1 = strtok(temp, "->");
      token2 = strtok(NULL, "->");
      strcpy(rules[i].left, token1);
      strcpy(rules[i].right, token2);
   }
   printf("\nEnter the Input String: ");
```

```
scanf("%s", input);
printf("\n--------------------------\n");
printf("STACK\t INPUT\tAction");
printf("\n--------------------------\n");
i=0;
while (1)
{
   if (i < strlen(input))
   {
      ch[0] = input[i];
      ch[1] = '\0';
      i++;
      strcat(stack, ch);
      printf("\n%s\t", stack);
      for (int k = i; k < strlen(input); k++)
      {
         printf("%c", input[k]);
      }
      printf("\tShift %s\n", ch);
   }
   for (j = 0; j < rule_count; j++)
   {
      substring = strstr(stack, rules[j].right);
      if (substring != NULL)
      {
         stack_length = strlen(stack);
         substring_length = strlen(substring);
         stack_top = stack_length - substring_length;
         stack[stack_top] = '\0';
         strcat(stack, rules[j].left);
         printf("%s\t", stack);
         for (int k = i; k < strlen(input); k++)
         {
            printf("%c", input[k]);
         }
         printf("\tReduce %s->%s\n", rules[j].left, rules[j].right);
         j=-1;
```

```
            }
          }
          if (strcmp(stack, rules[0].left) == 0 && i == strlen(input))
          {
              printf("\nAccepted");
              flag=1;
              break;
          }
        else if(input[i]=='\0')
        {
            break;
        }

      }
      if(flag==0)
      {
          printf("\nrejected");
      }
  }
```

9. **Write a C program to generate three address code.**

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define MAX_SIZE 100
typedef struct
{
int top;
char items[MAX_SIZE];
} Stack;
void push(Stack *s, char c);
char pop(Stack *s);
int precedence(char c);
void infixToPostfix(char *expression, char *postfix);
void generateThreeAddressCode(char *postfix);
int main()
{
char expression[MAX_SIZE];
char postfix[MAX_SIZE];
char name[20];
```

```
char rollno[20];
printf("Enter an arithmetic expression: ");
fgets(expression, sizeof(expression), stdin);
printf("\nenter name of the student:");
scanf("%s",name);
printf("\nenter rollno of the student:");
scanf("%s",rollno);
printf("\nthe name of the student  is :%s",name);
printf("\n\nthe rollno of the student is :%s \n",rollno);
expression[strcspn(expression, "\n")] = '\0';
infixToPostfix(expression, postfix);
generateThreeAddressCode(postfix);
return 0;
}
void push(Stack *s, char c)
{
 if (s->top == MAX_SIZE - 1)
 {
   printf("Stack Overflow\n");
   exit(1);
 }
s->items[++(s->top)] = c;
}
char pop(Stack *s)
{
 if (s->top == -1)
 {
   printf("Stack Underflow\n");
   exit(1);
 }
return s->items[(s->top)--];
}
int precedence(char c)
{
 if (c == '+' || c == '-')
 {
  return 1;
 }
else
if (c == '*' || c == '/')
 {
  return 2;
 }
else
```

```
{
return 0;
}
}
void infixToPostfix(char *expression, char *postfix)
{
 Stack stack;
 stack.top = -1;
 int i = 0, j = 0;
 while (expression[i] != '\0')
 {
 char c = expression[i];
 if (isalpha(c))
 {
  postfix[j++] = c;
 }
 else if (c == '(')
 {
 push(&stack, c);
 }
else if (c == ')')
{
while(stack.items[stack.top]!='(')
{
postfix[j++]=pop(&stack);
}
pop(&stack);
}
else
{
 while (stack.top != -1 && precedence(stack.items[stack.top]) >= precedence(c))
 {
 postfix[j++] = pop(&stack);
 }
 push(&stack,c);
 }
i++;
}
while(stack.top!=-1)
{
postfix[j++]=pop(&stack);
}
postfix[j]='\0';
}
```

```
void generateThreeAddressCode(char *postfix)
{
Stack stack;
stack.top = -1;
int temp_count = 1;
int i = 0;
printf("Three Address Code:\n");
while (postfix[i] != '\0')
{
 char c = postfix[i];
 if (isalpha(c))
 {
  push(&stack, c);
 }
 else
 {
  char operand2 = pop(&stack);
  char operand1 = pop(&stack);
  char temp_var[10];
  sprintf(temp_var,"temp%d",temp_count++);
  if(isdigit(operand2) && isdigit(operand1))
  {
      printf("%s = temp%c %c temp%c\n", temp_var, operand1, c, operand2);
      push(&stack, temp_var[4]);
  }
  else if(isdigit(operand1))
  {
     printf("%s = temp%c %c %c\n", temp_var, operand1, c, operand2);
      push(&stack, temp_var[4]);
  }
  else if(isdigit(operand2))
  {
     printf("%s = %c %c temp%c\n", temp_var, operand1, c, operand2);
      push(&stack, temp_var[4]);
  }
  else
  {
     printf("%s = %c %c %c\n", temp_var, operand1, c, operand2);
     push(&stack, temp_var[4]);
  }
 }
i++;
}
}
```

10. Design a LALR bottom up parser for the given language.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
void push(char *,int *,char);
char stacktop(char *);
void isproduct(char,char);
int ister(char);
int isnter(char);
int isstate(char);
void error();
void isreduce(char,char);
char pop(char *,int *);
void printt(char *,int *,char [],int);
void rep(char [],int);
struct action
{
char row[5][7];
};
struct action A[11]={
{"emp","sd","emp","se","emp"},
{"emp","emp","emp","emp","acc"},
{"sf","emp","emp","emp","emp"},
{"emp","emp","emp","sh","emp"},
{"rf","emp","si","emp","emp"},
{"emp","emp","emp","emp","rb"},
{"emp","emp","sj","emp","emp"},
{"sk","emp","rf","emp","emp"},
{"emp","emp","emp","emp","rd"},
{"emp","emp","emp","emp","rc"},
{"emp","emp","emp","emp","rb"},
};
struct goto_t
{
char r[2][7];
};
struct goto_t G[11]={
{"b","c"},
{"emp","emp"},
{"emp","emp"},
{"emp","g"},
{"emp","emp"},
{"emp","emp"},
{"emp","emp"},
{"emp","emp"},
{"emp","emp"},
{"emp","emp"}
```

```
};
char ter[8]={'u','v','w','z','$'};
char nter[3]={'S','A'};
char states[15]={'a','b','c','d','e','f','g','h','i','j','k'};
char stack[100];
int top=-1;
char temp[20];
struct grammar
{
char left;
char right[5];
};
const struct grammar rl[5]={
{'S',"Au"},
{'S',"vAw"},
{'S',"zw"},
{'S',"vzu"},
{'A',"z"}
};
int main()
{
char name[20];
char rollno[20];
printf("\nEnter  the Name of the Student:");
scanf("%s",name);
printf("\nEnter the Roll Number of the Student:");
scanf("%s",rollno);
printf("\n Name of the Student is:%s",name);
printf("\n Roll no of the Student is:%s",rollno);
char inp[80],x,p,dl[80],y,bl='a';
int i=0,j,k,l,n,m,c,len;
printf(" \nEnter the String  be Parsed :");
scanf("%s",inp);
len=strlen(inp);
inp[len]='$';
inp[len+1]='\0';
push(stack,&top,bl);
printf("\n STACK \t\t\t INPUT");
printt(stack,&top,inp,i);
do
{
x=inp[i];
p=stacktop(stack);
isproduct(x,p);
if(strcmp(temp,"emp")==0)
error();
if(strcmp(temp,"acc")==0)
break;
```

```
else
{
if(temp[0]=='s')
{
push(stack,&top,inp[i]);
push(stack,&top,temp[1]);
i++;
}
else
{
if(temp[0]=='r')
{
j=isstate(temp[1]);
strcpy(temp,rl[j-2].right);
dl[0]=rl[j-2].left;
dl[1]='\0';
n=strlen(temp);
for(k=0;k<2*n;k++)
pop(stack,&top);
for(m=0;dl[m]!='\0';m++)
push(stack,&top,dl[m]);
l=top;
y=stack[l-1];
isreduce(y,dl[0]);
for(m=0;temp[m]!='\0';m++)
push(stack,&top,temp[m]);
}
}
}
printt(stack,&top,inp,i);
}while(inp[i]!='\0');
if(strcmp(temp,"acc")==0)
printf(" \n The Given Input String is Successfully Parsed ");
else
printf(" \n Do Not Accept the Input String ");
getch();
}
void push(char *s,int *sp,char item)
{
if(*sp==100)
printf(" Stack is Full ");
else
{
*sp=*sp+1; s[*sp]=item;
}
}
char stacktop(char *s)
{
```

```
char i;
i=s[top];
return i;
}
void isproduct(char x,char p)
{
int k,l;
k=ister(x);
l=isstate(p);
strcpy(temp,A[l-1].row[k-1]);
}
int ister(char x)
{
int i;
for(i=0;i<8;i++)
if(x==ter[i])
return i+1;
return 0;
}
int isnter(char x)
{
int i;
for(i=0;i<3;i++)
if(x==nter[i])
return i+1;
return 0;
}
int isstate(char p)
{
int i;
for(i=0;i<15;i++)
if(p==states[i]) return i+1;
return 0;
}
void error()
{
printf(" Unable to Parse the the Input string ");
exit(0);
}
void isreduce(char x,char p)
{
int k,l;
k=isstate(x);
l=isnter(p);
strcpy(temp,G[k-1].r[l-1]);
}
char pop(char *s,int *sp)
{
```

```
char item;
if(*sp==-1)
printf(" Stack is Empty ");
else
{
item=s[*sp];
*sp=*sp-1;
}
return item;
}
void printt(char *t,int *p,char inp[],int i)
{
int r;
printf("\n");
for(r=0;r<=*p;r++)
rep(t,r);
printf("\t\t\t");
for(r=i;inp[r]!='\0';r++)
printf("%c",inp[r]);
}
void rep(char t[],int r)
{
char c;
c=t[r];
switch(c)
{
case 'a':
printf("0");
break;
case 'b':
printf("1");
break;
case 'c':
printf("2");
break;
case 'd':
printf("3");
break;
case 'e':
printf("4");
break;
case 'f':
printf("5");
break;
case 'g':
printf("6");
break;
case 'h':
```

```
        printf("7");
        break;
        case 'i':
        printf("8");
        break;
        case 'j':
        printf("9");
        break;
        case 'k':
        printf("10");
        break;
        default :
        printf("%c",t[r]);
        break;
        }
        }
```

11.Convert the BNF rules into YACC form and write code to generate abstract syntax tree.

Capital.l

```
%{
#include<stdio.h>
%}
%%
[A-Z]+[ \t\n] {printf("%s\n", yytext);}
. ;
%%

int main()
{
        printf("Enter the Input String: \n");
    yylex();
}

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

Keyword.l

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

%%
if |
else |
printf {printf("\n%s is a keyword",yytext);}
```

```
[0-9]+ {printf("\n%s is a number",yytext);}
[a-zA-z]+ {printf("\n%s is a word",yytext);}
.|\n {ECHO;}
%%

int main()
{
          printf("\nEnter the Input String: ");
          yylex();
}

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

Decimal.l

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

%%

if    { printf("\n%s is a keyword", yytext); }
else  { printf("\n%s is a keyword", yytext); }
printf { printf("\n%s is a keyword", yytext); }
[0-9]+ { printf("\n%s is a number", yytext); }
[a-zA-Z]+ { printf("\n%s is a word", yytext); }
[0-9]+"."[0-9]+ { printf("\n%s is a decimal number", yytext); }
.|\n { ECHO; }

%%

int main()
{
   printf("\nEnter the Input String: ");
   yylex();
}

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

12. A program to generate machine code from the abstract syntax tree generated by the parser.

```
#include<stdio.h>
#include<string.h>
int main()
{
char name[20];
char rollno[20];
printf("\nEnter  the Name of the Student:");
scanf("%s",name);
printf("\nEnter the Roll Number of the Student:");
scanf("%s",rollno);
printf("\nName of the Student is:%s",name);
printf("\nRoll no of the Student is:%s",rollno);
char inp[100][100];
int n,i,j,len;
int reg = 1;
printf("\nEnter the No of Statements:");
scanf("%d",&n);
for(i = 0; i < n; i++)
scanf("%s",&inp[i]);
printf("\nTARGET CODE");
printf("\n--------------------------------------");
for(i = 0; i < n; i++)
{
len = strlen(inp[i]);
for(j=2; j < len; j++)
{
if(inp[i][j] >= 97 && inp[i][j] <= 122)
{
printf("\nMOV R%d %c \n",reg++,inp[i][j]);
}
if(j == len-1 && inp[i][len-j] =='=')
{
j=3;
if(inp[i][j] == '+')
{
if(inp[i][4] >= 97 && inp[i][4] <= 122)
{
printf("\nADD R%d R%d\n",reg-2,reg-1);
printf("\nSTORE %c R%d\n",inp[i][0],reg-2);
}
else if(inp[i][4]>=48 && inp[i][4]<=57)
{
printf("\nADD R%d #%c\n",reg-1,inp[i][4]);
printf("\nSTORE %c R%d\n",inp[i][0],reg-1);
}
}
else if(inp[i][j]=='-')
{
if(inp[i][4] >= 97 && inp[i][4] <= 122)
{
printf("\nSUB R%d R%d\n",reg-2,reg-1);
```

```
printf("\nSTORE %c R%d\n",inp[i][0],reg-2);
}
else if(inp[i][4]>=48 && inp[i][4]<=57)
{
printf("\nSUB R%d #%c\n",reg-1,inp[i][4]);
printf("\nSTORE %c R%d\n",inp[i][0],reg-1);
}

}
else if(inp[i][j]=='*')
{
if(inp[i][4] >= 97 && inp[i][4] <= 122)
{
printf("\nMUL R%d R%d\n",reg-2,reg-1);
printf("\nSTORE %c R%d\n",inp[i][0],reg-2);
}
else if(inp[i][4]>=48 && inp[i][4]<=57)
{
printf("\nMUL R%d #%c\n",reg-1,inp[i][4]);
printf("\nSTORE %c R%d\n",inp[i][0],reg-1);
}

}
else if(inp[i][j]=='/')
{
if(inp[i][4] >= 97 && inp[i][4] <= 122)
{
printf("\nDIV R%d R%d\n",reg-2,reg-1);
printf("\nSTORE %c R%d\n",inp[i][0],reg-2);
}
else if(inp[i][4]>=48 && inp[i][4]<=57)
{
printf("\nDIV R%d #%c\n",reg-1,inp[i][4]);
printf("\nSTORE %c R%d\n",inp[i][0],reg-1);
}
}
else
{
printf("\nSTORE %c R%d\n",inp[i][0],reg-1);
}

break;
}
}
}
return 0;
}
```