3. Design, develop and implement YACC/C program to construct *Predictive / LL(1) ParsingTable* for the grammar rules: *A →aBa , B →bB | ε*. Use this table to parse the sentence: *abba$*.

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

void firstset();
void followset();
void parsingtable();
void parseinput();
void err();

int i, count, j, k, n, l[10],temp;
char grm[10][20], fst[10][20], fol[10][20], stk[20], matched[20], inpt[20], inp[20];

void main()
{
printf("The given grammar is\n A->aBa\n B->bB|@ \n\n");
printf("Enter the number of rules\n");
scanf("%d",&n);
printf("Enter the grammar and please enter @ instead of Epsilon\n");
for(i=0;i<n;i++)
        scanf("%s",grm[i]);
        firstset();
        followset();
        parsingtable();
        parseinput();
}

void firstset()
{
        printf("\nThe First Set is \n");
        for(i=0;i<n;i++)
        {
                printf("FIRST[%c]={",grm[i][0]);
                count=0;
                j=3;
                while(grm[i][j]!='\0')
                {
                        if(!((grm[i][j]>64)&&(grm[i][j]<91)))
                        {
                                fst[i][count]=grm[i][j];
                                printf("%c,",fst[i][count]);
                                count=count+1;
                        }
                        for(;grm[i][j]!='|'&&grm[i][j]!='\0';j++);
                        j=j+1;
                }
                printf("\b}\n");
        }
}

void followset()
{
        printf("\nThe Follow set is\n");
        for(k=0;k<n;k++)
        {
                count=0;
                printf("FOLLOW[%c]={",grm[k][0]);
                if(k==0)
                {
                        printf("$,");
                        fol[k][count]='$';
                        count=count+1;
                }
                for(i=0;i<n;i++)
                {
                        for(j=3;grm[i][j]!='\0';j++)
                        {

        if((grm[i][j]==grm[k][0])&&(grm[i][j+1]!='\0'&&grm[i][j+1]!='|'))
                                {
                                        if(!((grm[i][j+1]>64)&&(grm[i][j+1]<91)))
                                        {
                                        fol[k][count]=grm[i][j+1];
                                                printf("%c,",fol[k][count]);
                                                count=count+1;
                                        }
                                }
                        }
                }
                printf("\b}\n");
        }
}

void parsingtable()
{
printf("\nThe Parsing Table is\n");
        char p[10]="A->aBa",q[10]="B->bB",r[10]="B->@",tble[2][4];
        tble[0][0]='A';
        tble[1][0]='B';
        int j,k;
        for(i=0;i<n;i++)
        {
                j=0;
                while(fst[i][j]!='\0')
                {
                        if(fst[i][j]=='a')
```

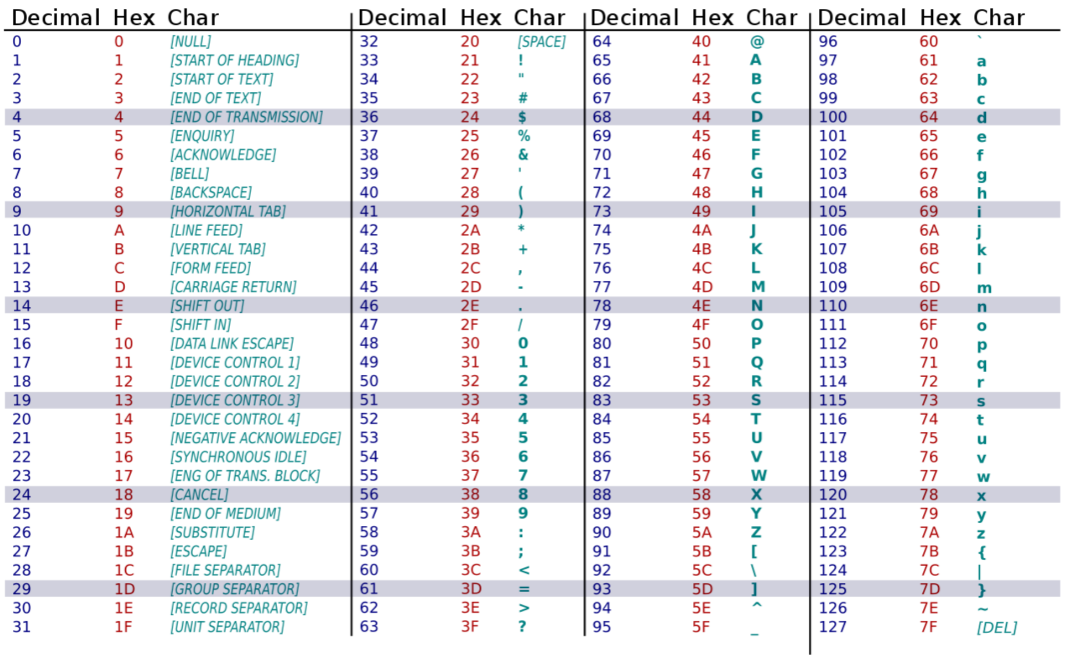
```
                                        tble[i][1]='p';
                                else if(fst[i][j]=='b')
                                        tble[i][2]='q';
                                else if(fst[i][j]=='@')
                                {
                                        k=0;
                                        for(k=0;fol[i][k]!='\0';k++)
                                        {
                                                if(fol[i][k]=='a')
                                                        tble[i][1]='r';
                                        }
                                }
                                j++;
                        }
                }
                printf("\ta\tb\t$\t\n");
                for(i=0;i<2;i++)
                {
                        for(j=0;j<4;j++)
                        {

                if(tble[i][j]!='p'&&tble[i][j]!='q'&&tble[i][j]!='r'&&tble[i][j]!='A'&&tble[i][j]!='B')
                                        tble[i][j]=' ';
                                if(tble[i][j]=='p')
                                        printf("%s\t",p);
                                else if(tble[i][j]=='q')
                                        printf("%s\t",q);
                                else if(tble[i][j]=='r')
                                        printf("%s\t",r);
                                else
                                        printf("%c\t",tble[i][j]);


                        }
                        printf("\n");
                }
        }

        void parseinput()
        {
        printf("\nEnter the input string\n");
                scanf("%s",&inp);
                strcpy(inpt,inp);
                strcat(inpt,"$");
                printf("matched\t\tstack\t\tinput\t\taction\n");
                strcpy(stk,"A$");
                i=0;
                j=0;
                k=0;
                while(matched!=inp)
                {
                if(stk[i]==inpt[j])
                   {
                if(stk[i]=='$')
                      {
                printf("%s\t\t%s\t\t%s\t\t accepted\n",matched,stk,inpt);
                break;
                      }
                temp=stk[i];
                printf("%s\t\t%s\t\t%s\t\t pop %c\n",matched,stk,inpt,temp);

                stk[i]=inpt[j]=' ';
                      i=i+1;
                      j=j+1;
                matched[k]=temp;
                           k=k+1;
                   }
                        else if(stk[i]=='A'&&inpt[j]=='a')
                        {
                                i=0;

                                printf("%s\t\t%s\t\t%s\t\t A->aBa\n",matched,stk,inpt);
                                strcpy(stk,"aBa$");
                        }
                        else if(stk[i]=='B')
                        {
                                i=0;
                                if(inpt[j]=='b')
                                {

                                        printf("%s\t\t%s\t\t%s\t\t B->bB\n",matched,stk,inpt);
                                        strcpy(stk,"bBa$");
                                }
                                else if(inpt[j]=='a')
                                {

                                        printf("%s\t\t%s\t\t%s\t\t B->@\n",matched,stk,inpt);
                                        strcpy(stk,"a$");
                                }
                        }
                        else if(stk[i]='$'&&inpt[j]=='$')
                                break;
                        else
                                err();
                }
        }
        void err()
        {
        printf("%s\t\t%s\t\t%s\t\t error\n",matched,stk,inpt);
        exit(0);
```

```
}
```

/* **OUTPUT**

```
[root@localhost ss]# cc 3.c
[root@localhost ss]# ./a.out
The given grammar is
 A->aBa
 B->bB|@

Enter the number of rules
2
Enter the grammar and please enter @ instead of Epsilon
A->aBa
B->bB|@

The First Set is
FIRST[A]={a}
FIRST[B]={b,@}

The Follow set is
FOLLOW[A]={$}
FOLLOW[B]={a}

The Parsing Table is
        a          b                $
A   A->aBa
B    B->@       B-> bB

Enter the input string
abba
matched              stack          input          action
                     A$             abba$           A->aBa
                     aBa$           abba$           pop a
a                    Ba$             bba$           B->bB
a                    bBa$            bba$           pop b
ab                   Ba$              ba$           B->bB
ab                   bBa$             ba$           pop b
abb                   Ba$              a$           B->@
abb                  a$                a$           pop a
abba                  $                 $           accepted


[root@localhost ss]# ./a.out
The given grammar is
 A->aBa
 B->bB|@

Enter the number of rules
```

```
2


Enter the grammar and please enter @ instead of Epsilon
A->aBa
B->bB|@

The First Set is
FIRST[A]={a}
FIRST[B]={b,@}

The Follow set is
FOLLOW[A]={$}
FOLLOW[B]={a}

The Parsing Table is
a               b               $
A                               A->aBa
B               B->@            B->bB

Enter the input string
abb
matched                 stack           input           action
                        A$              abb$             A->aBa
                        aBa$            abb$             pop a
a                        Ba$             bb$             B->bB
a                       bBa$             bb$             pop b
ab                       Ba$              b$             B->bB
ab                      bBa$              b$             pop b
abb                      Ba$                $               B->@
abb                     a$                 $             Error
*/
```



Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

4. Design, develop and implement YACC/C program to demonstrate *Shift Reduce Parsing* technique for the grammar rules: *E →E+T* | *T, T →T*F* | *F, F →(E)* | *id* and parse the sentence: *id* + *id* * *id*.

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

int len,top=-1,i,j;
char str[20],stk[20];
void errfcn();
void stkfcn();
void chk();
void smch();

void main()
{
        puts("Note:  Do Not give spaces in between the operator in the input\n");
        puts("The given GRAMMAR is \nE->E+T|T\nT->T*F|F\nF->(E)|id\n");
        puts("enter the input srting");
        gets(str);
        len=strlen(str);
        puts("stack\t\tinput\t\taction\n");
        printf("$%s\t\t%s$\n",stk,str);
        for(i=0;i<len;i++)
        {
                if(str[i]=='i' && str[i+1]=='d')
                {
                        str[i]=str[i+1]=' ';
                        top=top+1;
                        printf("$%sid\t\t%s$\t\tSHIFT->id\n",stk,str);
                        stk[top]='F';
                        printf("$%s\t\t%s$\t\tREDUCE to F->id\n",stk,str);
                        stkfcn();
                        smch();
                        i=i+1;
                }
                else if(str[i]=='+' || str[i]=='*')
                {
                        top=top+1;
                        stk[top]=str[i];
                        str[i]=' ';
                        if(stk[top]=='+')
                        printf("$%s\t\t%s$\t\tSHIFT->+\n",stk,str);
                        else
                        printf("$%s\t\t%s$\t\tSHIFT->*\n",stk,str);

                        if((stk[0]=='+'|| stk[0]=='*')||((stk[top]=='+'|| stk[top]=='*')&&(stk[top-
1]=='+'|| stk[top-1]=='*')))
                    errfcn();
                }
                else
                        errfcn();
        }
        if(stk[top]=='+'||stk[top]=='*')
        errfcn();
        chk();
        if(top==0)
        {
        if(stk[top]=='F')
           {
        stk[top]='T';
                printf("$%s\t\t%s$\t\tREDUCE to T->F\n",stk,str);
           }
                if(stk[top]=='T')
                {
                stk[top]='E';
                printf("$%s\t\t%s$\t\tREDUCE to E->T\n",stk,str);
                }
        }
        printf("$%s\t\t%s$\t\tSUCCESS\n",stk,str);
}

void stkfcn()
{
        if((top==0)&&((str[i+2]=='+')||(str[i+2]==' ')))
        {
                stk[top]='T';
                printf("$%s\t\t%s$\t\tREDUCE to T->F\n",stk,str);
```

```
                stk[top]='E';
                printf("$%s\t\t%s$\t\tREDUCE to E->T\n",stk,str);
                return;
        }
        else if((top==0)&&(str[i+2]=='*'))
        {
                stk[top]='T';
                printf("$%s\t\t%s$\t\tREDUCE to T->F\n",stk,str);
                return;
        }
        if(stk[top-1]=='+')
        {
                stk[top]='T';
                printf("$%s\t\t%s$\t\tREDUCE to T->F\n",stk,str);
                chk();
        }
        else if(stk[top-1]=='*')
                chk();
}

void chk()
{
        if(stk[top-1]=='*')
        {
                stk[top-1]=stk[top]=' ';
                top=top-2;
                printf("$%s\t\t%s$\t\tREDUCE to T->T*F\n",stk,str);
                if((str[i+2]=='+')&&(top==0))
                {
                        stk[top]='E';
                        printf("$%s\t\t%s$\t\tREDUCE to E->T\n",stk,str);
                }
        }
        else if((stk[top-1]=='+' && str[i+2]=='+')||(stk[top-1]=='+' && str[i+2]!='*'))
        {
        if(top-2==0)
           {
                        stk[top-2]='E';
           }
        else
           {
        stk[0]='E';
        for(j=3;j<=top;j++)
              {
        stk[j]=stk[j-2];
              }
           }
        stk[top-1]=stk[top]=' ';
        top=top-2;
        printf("$%s\t\t%s$\t\tREDUCE to E->E+T\n",stk,str);
```

```
          }
}

void smch()
{
if(stk[top-1]=='+' && str[i+2]=='+')
    {
if(top-2==0)
             {
                              stk[top-2]='E';
             }
          else
             {
          stk[0]='E';
          for(j=3;j<=top;j++)
                {
          stk[j]=stk[j-2];
                }
             }
          stk[top-1]=stk[top]=' ';
          top=top-2;
          printf("$%s\t\t%s$\t\tREDUCE to E->T\n",stk,str);
      }
}


void errfcn()
{
          printf("ERROR:invalid argument\n");
          exit(0);
}
```

/* **OUTPUT**

Note: Do Not give spaces in between the operator in the input

```
[root@localhost ss]# cc 4.c
[root@localhost ss]# ./a.out
The given GRAMMAR is
E->E+T|T
T->T*F|F
F->(E)|id

enter the input srting
id+id*id
stack           input                   action

$               id+id*id$
$id               +id*id$               SHIFT->id
$F                +id*id$               REDUCE to F->id
```

```
$T              +id*id$             REDUCE to T->F
$E              +id*id$             REDUCE to E->T
$E+              id*id$             SHIFT->+
$E+id              *id$             SHIFT->id
$E+F               *id$             REDUCE to F->id
$E+T               *id$             REDUCE to T->F
$E+T*               id$             SHIFT->*
$E+T*id               $             SHIFT->id
$E+T*F                $             REDUCE to F->id
$E+T                  $             REDUCE to T->T*F
$E                    $             REDUCE to E->E+T
$E                    $             SUCCESS

[root@localhost ss]# ./a.out
```
Note: Do Not give spaces in between the operator in the input

```
The given GRAMMAR is
E->E+T|T
T->T*F|F
F->(E)|id
enter the input srting
id++id
stack           input           action

$               id++id$
$id               ++id$         SHIFT->id
$F                ++id$         REDUCE to F->id
$T                ++id$         REDUCE to T->F
$E                ++id$         REDUCE to E->T
$E+                +id$         SHIFT->+
$E++                id$         SHIFT->+
ERROR: invalid argument

*/
```