```
                point1++;
        }
}

void follow(char c)
{
        int i, j;
        if (production[0][0] == c) {
                f[m++] = '$';
        }
        for (i = 0; i < 10; i++) {
                for (j = 2; j < 10; j++) {
                        if (production[i][j] == c) {
                                if (production[i][j + 1] != '\0') {
                                        followfirst(production[i][j + 1], i,(j + 2));
                                }
                                if (production[i][j + 1] == '\0'
                                        && c != production[i][0]) {
                                        follow(production[i][0]);
                                }
                        }
                }
        }
}

void findfirst(char c, int q1, int q2)
{
        int j;
        if (!(isupper(c))) {
                first[n++] = c;
        }
        for (j = 0; j < count; j++) {
                if (production[j][0] == c) {
                        if (production[j][2] == '#') {
                                if (production[q1][q2] == '\0')
```

```
r2038@administrator-PowerEdge-R820:~/CD/firstandfollow$ ./a.out
Enter the no: of expressions: 8
Enter expression 1: E=TR
Enter expression 2: R=+TR
Enter expression 3: R=#
Enter expression 4: T=FY
Enter expression 5: Y=*FY
Enter expression 6: Y=#
Enter expression 7: F=(E)
Enter expression 8: F=i

 First(E) = { (, i, }

 First(R) = { +, #, }

 First(T) = { (, i, }

 First(Y) = { *, #, }

 First(F) = { (, i, }

------------------------------------------------

 Follow(E) = { $, ),  }

 Follow(R) = { $, ),  }

 Follow(T) = { +, $, ),  }

 Follow(Y) = { +, $, ),  }

 Follow(F) = { *, +, $, ),  }
```

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

void followfirst(char, int, int);
void follow(char c);
void findfirst(char, int, int);
int count, n = 0,m=0,k,e;
char calc_first[10][100],calc_follow[10][100];
char production[10][10],f[10], first[10],ck;


int main(int argc, char** argv)
{
        int jm = 0,km = 0,i, choice;
        char c, ch;
        count = 8;
        printf("Enter the no: of expressions: ");
        scanf("%d",&count);
        for(i=0;i<count;i++)
        {
                printf("Enter expression %d: ",i+1);
                scanf("%s",production[i]);
        }
        int kay,ptr = -1;
        char done[count];
        for (k = 0; k < count; k++) {
                for (kay = 0; kay < 100; kay++) {
                        calc_first[k][kay] = '!';
                }
        }
        int point1 = 0, point2, xxx;
                for (k = 0; k < count; k++) {
                c = production[k][0];
                point2 = 0;
                xxx = 0;
                for (kay = 0; kay <= ptr; kay++)
                        if (c == done[kay])
                                xxx = 1;
                if (xxx == 1)
                        continue;
                findfirst(c, 0, 0);
                ptr += 1;
                done[ptr] = c;
                printf("\n First(%c) = { ", c);
                calc_first[point1][point2++] = c;
                for (i = 0 + jm; i < n; i++) {
                        int lark = 0, chk = 0;
                        for (lark = 0; lark < point2; lark++) {
                                        if (first[i] == calc_first[point1][lark]) {
                                        chk = 1;
                                        break;
                                }
                        }
                        if (chk == 0) {
                                printf("%c, ", first[i]);
                                calc_first[point1][point2++] = first[i];
                        }
                }
                printf("}\n");
                jm = n;
                point1++;
        }
        printf("\n");
        printf("-----------------------------------------------""\n\n");
        char donee[count];
        ptr = -1;
        for (k = 0; k < count; k++) {
                for (kay = 0; kay < 100; kay++) {
```

```
                                first[n++] = '#';
                        else if (production[q1][q2] != '\0'
                                        && (q1 != 0 || q2 != 0)) {
                                findfirst(production[q1][q2], q1,
                                                (q2 + 1));
                        }
                        else
                                first[n++] = '#';
                }
                else if (!isupper(production[j][2])) {
                        first[n++] = production[j][2];
                }
                else {
                        findfirst(production[j][2], j, 3);
                }
        }
    }
}

void followfirst(char c, int c1, int c2)
{
        int k;
        if (!(isupper(c)))
                f[m++] = c;
        else {
                int i = 0, j = 1;
                for (i = 0; i < count; i++) {
                        if (calc_first[i][0] == c)
                                break;
                }
                while (calc_first[i][j] != '!') {
                        if (calc_first[i][j] != '#') {
                                f[m++] = calc_first[i][j];
                        }
                        else {
                                calc_follow[k][kay] = '!';
                        }
        }
        point1 = 0;
        int land = 0;
        for (e = 0; e < count; e++) {
                ck = production[e][0];
                point2 = 0;
                xxx = 0;
                for (kay = 0; kay <= ptr; kay++)
                        if (ck == donee[kay])
                                xxx = 1;
                if (xxx == 1)
                        continue;
                land += 1;
                follow(ck);
                ptr += 1;
                donee[ptr] = ck;
                printf(" Follow(%c) = { ", ck);
                calc_follow[point1][point2++] = ck;
                for (i = 0 + km; i < m; i++) {
                        int lark = 0, chk = 0;
                        for (lark = 0; lark < point2; lark++) {
                                if (f[i] == calc_follow[point1][lark]) {
                                        chk = 1;
                                        break;
                                }
                        }
                        if (chk == 0) {
                                printf("%c, ", f[i]);
                                calc_follow[point1][point2++] = f[i];
                        }
                }
                printf(" }\n\n");
                km = m;
```

```
                                if (production[c1][c2] == '\0') {
                                        follow(production[c1][0]);
                                }
                                else {
                                        followfirst(production[c1][c2], c1,c2 + 1);
                                }
                        }
                        j++;
                }
        }
}
```

```
#include<stdio.h>
#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
int main()
   {

      printf("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id\n");
      printf("enter input string ");
      scanf("%s",a);
      c=strlen(a);
      strcpy(act,"SHIFT->");
      puts("stack \t input \t action");
      for(k=0,i=0; j<c; k++,i++,j++)
       {
         if(a[j]=='i' && a[j+1]=='d')
           {
              stk[i]=a[j];
              stk[i+1]=a[j+1];
              stk[i+2]='\0';
              a[j]=' ';
              a[j+1]=' ';
              printf("\n$%s\t%s$\t%sid",stk,a,act);
              check();
           }
         else
           {
              stk[i]=a[j];
              stk[i+1]='\0';
              a[j]=' ';
              printf("\n$%s\t%s$\t%ssymbols",stk,a,act);
              check();
           }
       }
        if(stk[0]=='E' && stk[1]=='\0')
        {
                printf("\nAccepted\n");
        }
```

```
        else
        {
                printf("\nrejected\n");
        }

   }
void check()
   {
     strcpy(ac,"REDUCE TO E");
     for(z=0; z<c; z++)
       if(stk[z]=='i' && stk[z+1]=='d')
         {
           stk[z]='E';
           stk[z+1]='\0';
           printf("\n$%s\t%s$\t%s",stk,a,ac);
           j++;
                check();
         }
     for(z=0; z<c; z++)
       if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
         {
           stk[z]='E';
           stk[z+1]='\0';
           stk[z+2]='\0';
           printf("\n$%s\t%s$\t%s",stk,a,ac);
           i=i-2;
                check();
         }
     for(z=0; z<c; z++)
       if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
         {
           stk[z]='E';
           stk[z+1]='\0';
           stk[z+1]='\0';
           printf("\n$%s\t%s$\t%s",stk,a,ac);
           i=i-2;
                check();
         }
     for(z=0; z<c; z++)
       if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
```

```
        {
          stk[z]='E';
          stk[z+1]='\0';
          stk[z+2]='\0';
          printf("\n$%s\t%s$\t%s",stk,a,ac);
          i=i-2;
               check();
        }
    for(z=0; z<c; z++)
      if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
        {
          stk[z]='E';
          stk[z+1]='\0';
          stk[z+1]='\0';
          printf("\n$%s\t%s$\t%s",stk,a,ac);
          i=i-2;
               check();
        }
    for(z=0; z<c; z++)
      if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')

        {
          stk[z]='E';
          stk[z+1]='\0';
          stk[z+1]='\0';
          printf("\n$%s\t%s$\t%s",stk,a,ac);
          i=i-2;
               check();
        }
  }
```

```
Acceptedr2038@administrator-PowerEdge-R820:~/CD$ nano srp1.c
r2038@administrator-PowerEdge-R820:~/CD$ gcc srp1.c
r2038@administrator-PowerEdge-R820:~/CD$ ./a.out
GRAMMAR is E->E+E
 E->E*E
 E->(E)
 E->id
enter input string id*id+id
stack    input   action

$id       *id+id$       SHIFT->id
$E        *id+id$       REDUCE TO E
$E*        id+id$       SHIFT->symbols
$E*id        +id$       SHIFT->id
$E*E         +id$       REDUCE TO E
$E           +id$       REDUCE TO E
$E+           id$       SHIFT->symbols
$E+id           $       SHIFT->id
$E+E            $       REDUCE TO E
$E              $       REDUCE TO E
Accepted
```