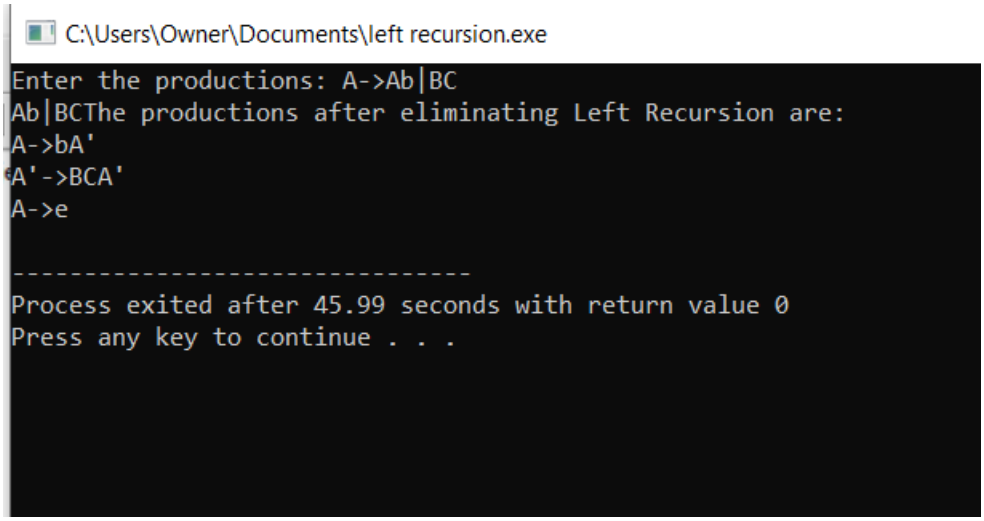
1.) left recursion :

```
#include<stdio.h>
#include<string.h>
int main()  {
   char input[100],l[50],r[50],temp[10],tempprod[20],productions[25][50];
   int i=0,j=0,flag=0,consumed=0;
   printf("Enter the productions: ");
   scanf("%1s->%s",l,r);
   printf("%s",r);
   while(sscanf(r+consumed,"%[^|]s",temp) == 1 && consumed <= strlen(r))  {
      if(temp[0] == l[0])  {
         flag = 1;
         sprintf(productions[i++],"%s->%s%s'\0",l,temp+1,l);
      }
      else
         sprintf(productions[i++],"%s'->%s%s'\0",l,temp,l);
      consumed += strlen(temp)+1;
   }
   if(flag == 1)  {
      sprintf(productions[i++],"%s->e\0",l);
      printf("The productions after eliminating Left Recursion are:\n");
      for(j=0;j<i;j++)
         printf("%s\n",productions[j]);
   }
   else
      printf("The Given Grammar has no Left Recursion");
```

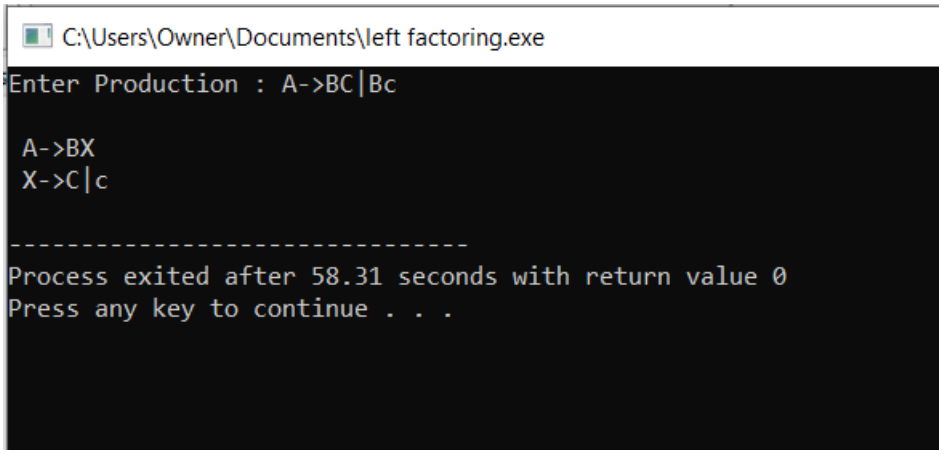



```
}
```

2.) left factoring output:

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

```
#include<string.h>
int main()
{
  char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
  int i,j=0,k=0,l=0,pos;
  printf("Enter Production : A->");
  gets(gram);
  for(i=0;gram[i]!='|';i++,j++)
     part1[j]=gram[i];
   part1[j]='\0';
   for(j=++i,i=0;gram[j]!='\0';j++,i++)
     part2[i]=gram[j];
   part2[i]='\0';
   for(i=0;i<strlen(part1)||i<strlen(part2);i++)
   {
     if(part1[i]==part2[i])
     {
        modifiedGram[k]=part1[i];
        k++;
        pos=i+1;
     }
   }
   for(i=pos,j=0;part1[i]!='\0';i++,j++){
     newGram[j]=part1[i];
   }
   newGram[j++]='|';
   for(i=pos;part2[i]!='\0';i++,j++){
     newGram[j]=part2[i];
   }
   modifiedGram[k]='X';
   modifiedGram[++k]='\0';
   newGram[j]='\0';
   printf("\n A->%s",modifiedGram);
   printf("\n X->%s\n",newGram);
}
```



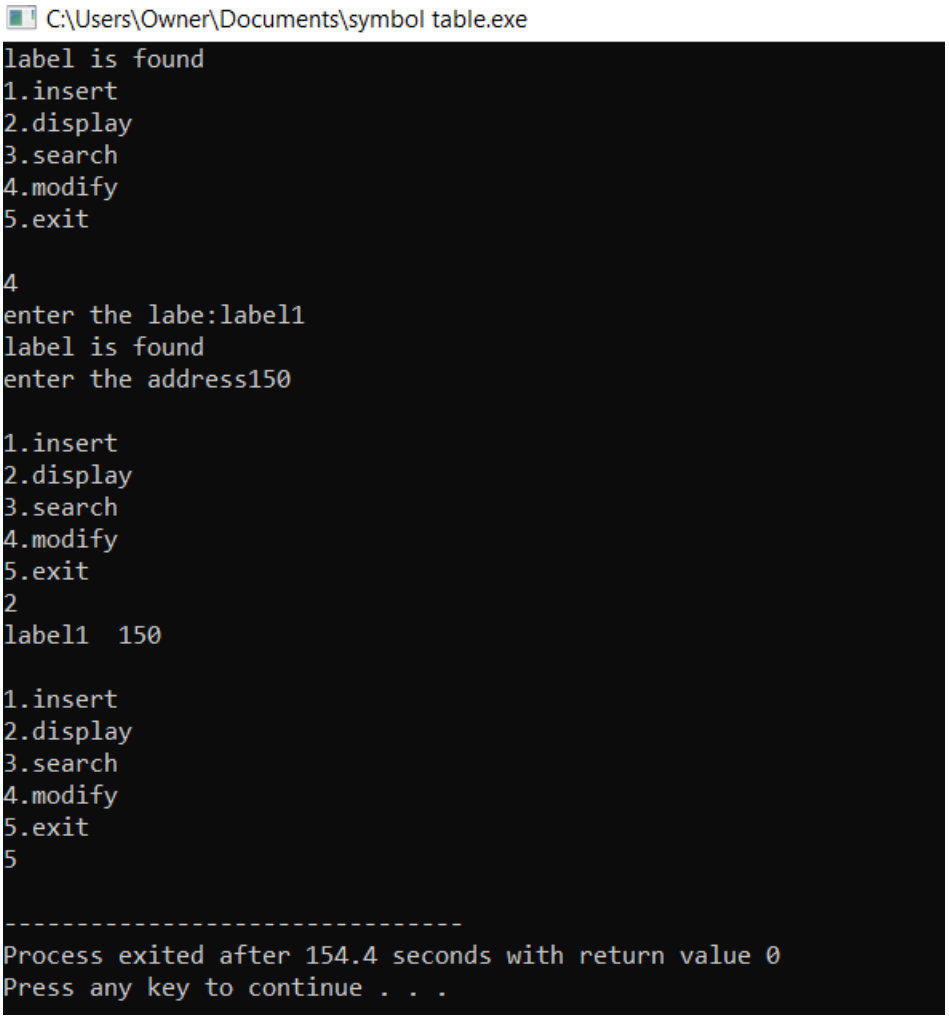
3.)symbol table:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int cnt=0;
struct symtab
{
    char label[20];
    int addr;
}
sy[50];
void insert();
int search(char *);
void display();
void modify();
int main()
{
int ch,val;
char lab[10];
do
{
    printf("\n1.insert\n2.display\n3.search\n4.modify\n5.exit\n");
```

```
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:
                insert();
                 break;
                case 2:
                        display();
                        break;
        case 3:
printf("enter the label");
                scanf("%s",lab);
                val=search(lab);
                if(val==1)
                printf("label is found");
                else
                printf("label is not found");
        break;
    case 4:
                modify();
        break;
    case 5:
                exit(0);
                break;
        }
    }while(ch<5);
}
void insert()
{
int val;
```

```
    char lab[10];
    int symbol;
    printf("enter the label");
    scanf("%s",lab);
    val=search(lab);
    if(val==1)
    printf("duplicate symbol");
    else
    {
        strcpy(sy[cnt].label,lab);
        printf("enter the address");
        scanf("%d",&sy[cnt].addr);
        cnt++;
    }
}
int search(char *s)
{
    int flag=0,i; for(i=0;i<cnt;i++)
    {
        if(strcmp(sy[i].label,s)==0)
        flag=1;
    }
return flag;
}
void modify()
{
    int val,ad,i;
    char lab[10];
    printf("enter the labe:");
    scanf("%s",lab);
```

```
    val=search(lab);
    if(val==0)
    printf("no such symbol");
    else
    {
        printf("label is found \n");
        printf("enter the address");
        scanf("%d",&ad);
        for(i=0;i<cnt;i++)
        {
                if(strcmp(sy[i].label,lab)==0)
                sy[i].addr=ad;
        }
    }
}
void display()
{
    int i;
    for(i=0;i<cnt;i++)
    printf("%s\t%d\n",sy[i].label,sy[i].addr);
    }
```

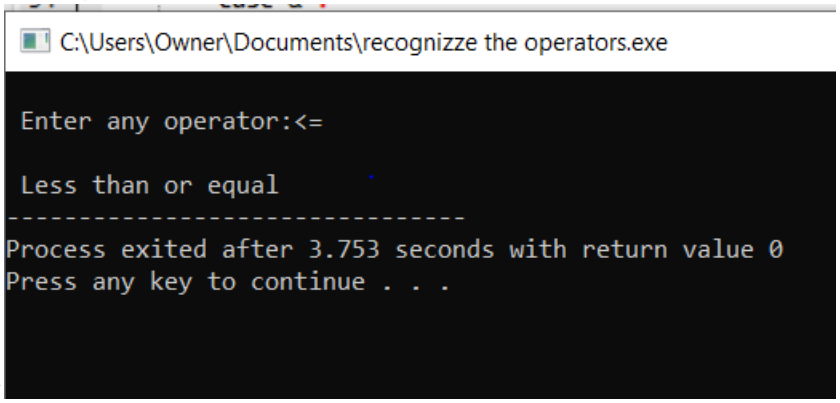


4.) recognize operators:

```
#include<stdio.h>
#include<conio.h>
int main()
{
	char s[5];
	printf("\n Enter any operator:");
	gets(s);
	switch(s[0])
	{
		case'>':
			if(s[1]=='=')
				printf("\n Greater than or equal");
			else
				printf("\n Greater than");
```

```
        break;
case'<':
        if(s[1]=='=')
                printf("\n Less than or equal");
        else
                printf("\nLess than");
        break;
case'=':
        if(s[1]=='=')
                printf("\nEqual to");
        else
                printf("\nAssignment");
        break;
case'!':
        if(s[1]=='=')
                printf("\nNot Equal");
        else
                printf("\n Bit Not");
        break;
case'&':
        if(s[1]=='&')
                printf("\nLogical AND");
        else
                printf("\n Bitwise AND");
        break;
case'|':
        if(s[1]=='|')
                printf("\nLogical OR");
        else
                printf("\nBitwise OR");
        break;
case'+':
        printf("\n Addition");
        break;
case'-':
        printf("\nSubstraction");
        break;
case'*':
```

```
            printf("\nMultiplication");
            break;
        case'/':
            printf("\nDivision");
            break;
        case'%':
            printf("Modulus");
            break;
        default:
            printf("\n Not a operator");
    }
```



}

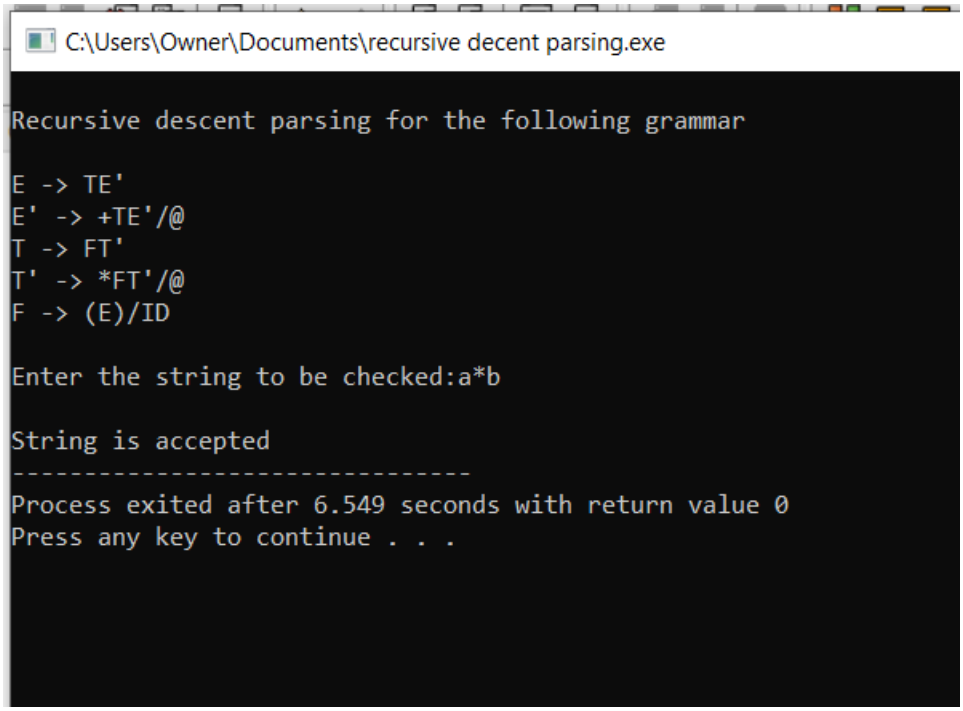
5.) recursive decent parsing:

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

char input[100];
int i;

int E();
int EP();
int T();
int TP();
int F();

int main(void) {
```

```
    printf("\nRecursive descent parsing for the following grammar\n");
    printf("\nE -> TE'\nE' -> +TE'/@\nT -> FT'\nT' -> *FT'/@\nF -> (E)/ID\n");
    printf("\nEnter the string to be checked:");
    fgets(input, sizeof(input), stdin);
    input[strcspn(input, "\n")] = '\0'; // Removing trailing newline

    i = 0; // Initialize index
    if (E()) {
        if (input[i] == '\0')
            printf("\nString is accepted");
        else
            printf("\nString is not accepted");
    } else
        printf("\nString not accepted");

    return 0;
}

int E() {
    if (T()) {
        if (EP())
            return 1;
        else
            return 0;
    } else
        return 0;
}

int EP() {
```

```
    if (input[i] == '+') {
        i++;
        if (T()) {
            if (EP())
                return 1;
            else
                return 0;
        } else
            return 0;
    } else
        return 1;
}

int T() {
    if (F()) {
        if (TP())
            return 1;
        else
            return 0;
    } else
        return 0;
}

int TP() {
    if (input[i] == '*') {
        i++;
        if (F()) {
            if (TP())
                return 1;
```

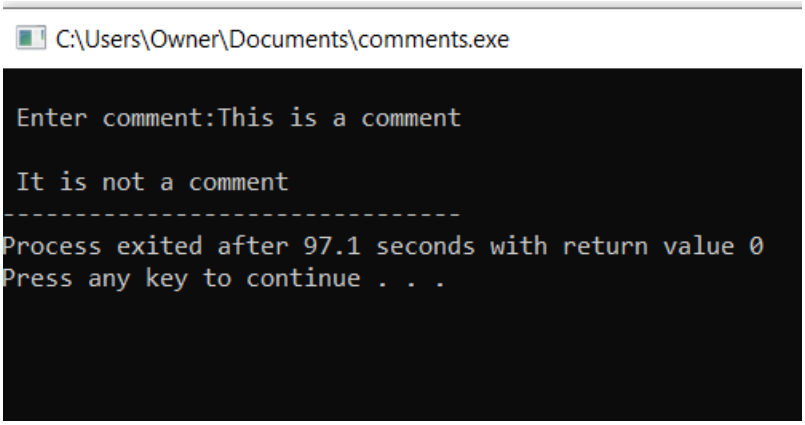
```
            else
                return 0;
        } else
            return 0;
    } else
        return 1;
}

int F() {
    if (input[i] == '(') {
        i++;
        if (E()) {
            if (input[i] == ')') {
                i++;
                return 1;
            } else
                return 0;
        } else
            return 0;
    } else if ((input[i] >= 'a' && input[i] <= 'z') || (input[i] >= 'A' && input[i] <= 'Z')) {
        i++;
        return 1;
    } else
        return 0;
    }
```



6.) comments:

```
#include<stdio.h>
#include<conio.h>
int main()
{
    char com[30];
    int i=2,a=0;
    printf("\n Enter comment:");
    gets(com);
    if(com[0]=='/')
    {
       if(com[1]=='/')
                printf("\n It is a comment");
       else if(com[1]=='*')
       {
                for(i=2;i<=30;i++)
                {
                        if(com[i]=='*'&&com[i+1]=='/')
```

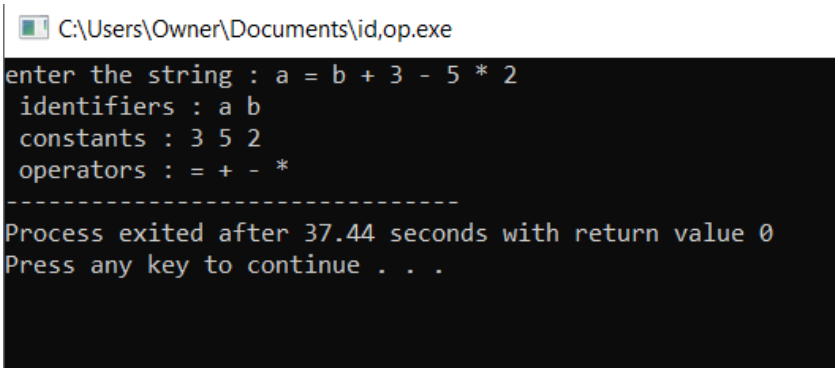
```
                {
                        printf("\n It is a comment");
                        a=1;
                        break;
                }
                else
                        continue;
        }
        if(a==0)
                printf("\n It is not a comment");
    }
    else
        printf("\n It is not a comment");
}
else
    printf("\n It is not a comment");
}
```



7.)id and operator output:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int main()
{
    int i,ic=0,m,cc=0,oc=0,j;
    char b[30],operators[30],identifiers[30],constants[30];
    printf("enter the string : ");
    scanf("%[^\n]s",&b);
    for(i=0;i<strlen(b);i++)
    {
    if(isspace(b[i]))
        {
       continue;
    }
    else if(isalpha(b[i]))
        {
        identifiers[ic] =b[i];
        ic++;
    }
    else if(isdigit(b[i]))
        {
        m=(b[i]-'0');
        i=i+1;
        while(isdigit(b[i]))
                {
                m=m*10 + (b[i]-'0');
                i++;
        }
        i=i-1;
        constants[cc]=m;
        cc++;
    }
    else
        {
        if(b[i]=='*')
                {
                operators[oc]='*';
                oc++;
        }
```

```
        else if(b[i]=='-')
                {
                operators[oc]='-';
                oc++;
        }
        else if(b[i]=='+')
                {
                operators[oc]='+';
                oc++;
        }
        else if(b[i]=='=')
                {
                operators[oc]='=';
                oc++;
        }
        }
  }
 printf(" identifiers : ");
  for(j=0;j<ic;j++)
   {
     printf("%c ",identifiers[j]);
  }
 printf("\n constants : ");
  for(j=0;j<cc;j++)
   {
     printf("%d ",constants[j]);
  }
 printf("\n operators : ");
   for(j=0;j<oc;j++)
    {
     printf("%c ",operators[j]);
   }
   }
```

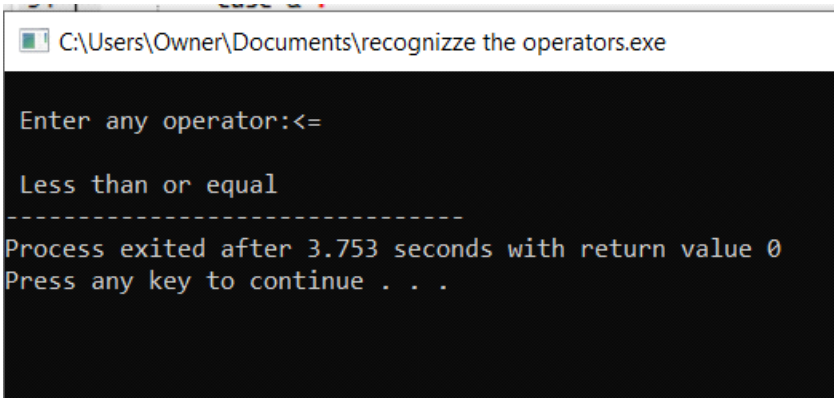


8.)In a class of Grade 3, Mathematics Teacher asked for the Acronym PEMDAS?. All of them are thinking for a while. A smart kid of the class Kishore of the class says it is Parentheses, Exponentiation, Multiplication,Division, Addition, Subtraction. Can you write a C Program to help the students to understand about the operator precedence parsing for an expression containing more than one operator, the order of evaluation depends on the order of operations.

```
#include<stdio.h>
#include<conio.h>   int
main()
{
        char s[5];
        printf("¥n Enter any operator:");
        gets(s);
        switch(s[0])
        {
                case'>':
                        if(s[1]=='=')
                                printf("¥n Greater than or equal");
                        else
                                printf("¥n Greater than");
                        break;
                case'<':
                        if(s[1]=='=')
                                printf("¥n Less than or equal");
                        else
```

```
                printf("¥nLess than");

        break;
case'=':
        if(s[1]=='=')
                printf("¥nEqual to");

        else
                printf("¥nAssignment");

        break;
case'!':
        if(s[1]=='=')
                printf("¥nNot Equal");

        else
            printf("¥n Bit Not");
        break;
case'&':
        if(s[1]=='&')
                printf("¥nLogical AND");

        else
                printf("¥n Bitwise AND");

        break;
case'|':
        if(s[1]=='|')
                printf("¥nLogical OR");

        else
                printf("¥nBitwise OR");

        break;
case'+':
        printf("¥n Addition");break;
```

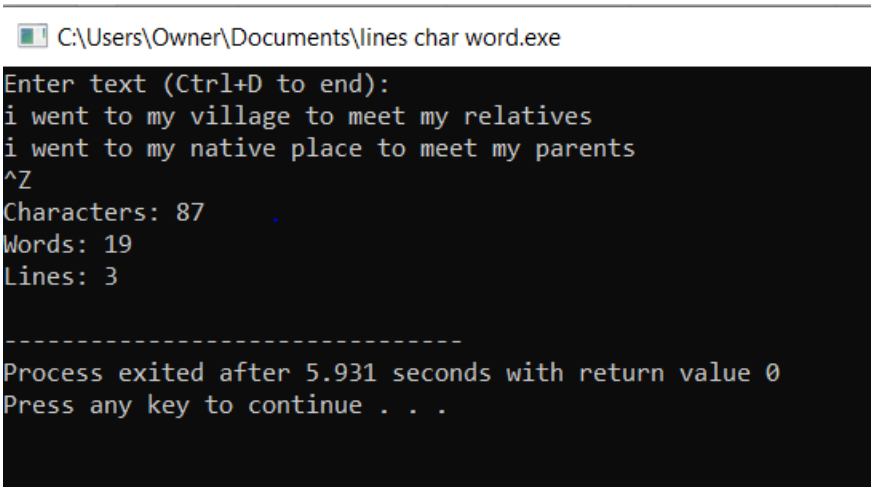
```
            case'-':
                printf("¥nSubstraction");break;
            case'*':
                printf("¥nMultiplication");break;
            case'/':
                printf("¥nDivision");
                break;
            case'%':
                printf("Modulus");break;
            default:
                printf("¥n Not a operator");
        }
}
```



9.)#include <stdio.h>

```
#include <ctype.h>

int main() {
    char ch;
    int charCount = 0, wordCount = 0, lineCount = 0;int inWord
    = 0;

    printf("Enter text (Ctrl+D to end):¥n");

    while ((ch = getchar()) != EOF) {
```

```
        charCount++;

        if (ch == '¥n') {
            lineCount++;
        }

        if (isspace(ch)) {
            inWord = 0;
        } else if (!inWord) {
            inWord = 1;
            wordCount++;
        }
    }

    // To account for the last line if it doesn't end with a newlineif (charCount
    > 0 && ch != '¥n') {
        lineCount++;

    }

    printf("Characters: %d¥n", charCount);
    printf("Words: %d¥n", wordCount); printf("Lines:
    %d¥n", lineCount);

    return 0;
}
```



10.)#include

<stdio.h>#include

<string.h> #include

<ctype.h> #include

<stdlib.h>

int tempVarCount = 0; // Counter for temporary variables

// Function to generate a new temporary

variablechar* newTemp() {

static char temp[5];

sprintf(temp, "t%d",

tempVarCount++);return temp;

}

// Function to print three-address code

void generateTAC(char* left, char op, char* right, char* result) {

printf("%s = %s %c %s¥n", result, left, op, right);

```
}

// Recursive function to parse the expression and generate
TACchar* parseExpression(char* expr, int start, int end) {
    int i, lastOp = -1, opPosition = -1, parentheses = 0;

    // Find the last operator in the expression that is outside of any
parentheses
    for (i = start; i <= end; i++)
        {if (expr[i] == '(') {
            parentheses++;
        } else if (expr[i] == ')')
            {parentheses--;
        } else if (parentheses == 0 && (expr[i] == '+' || expr[i] == '-
            ')) {lastOp = i;
        } else if (parentheses == 0 && (expr[i] == '*' || expr[i] == '/') &&
lastOp == -1) {
            opPosition = i;
        }
    }

    if (lastOp == -1) {
        lastOp = opPosition;
    }
    if (lastOp == -1) {
```

```
        if (expr[start] == '(' && expr[end] == ')') {
            return parseExpression(expr, start + 1, end - 1);
        } else {
            char* operand = (char*)malloc(2);
            operand[0] = expr[start];
            operand[1] = '¥0';
            return operand;
        }
    }

    char* left = parseExpression(expr, start, lastOp - 1);
    char* right = parseExpression(expr, lastOp + 1,
    end);char op = expr[lastOp];
    char* result = newTemp();

    generateTAC(left, op, right, result);

    return result;
}

int main() {
    char expr[100];

    printf("Enter an arithmetic expression: ");
    scanf("%s", expr);
```

```
    int len = strlen(expr);
    parseExpression(expr, 0, len -
    1);

    return 0;
}
```