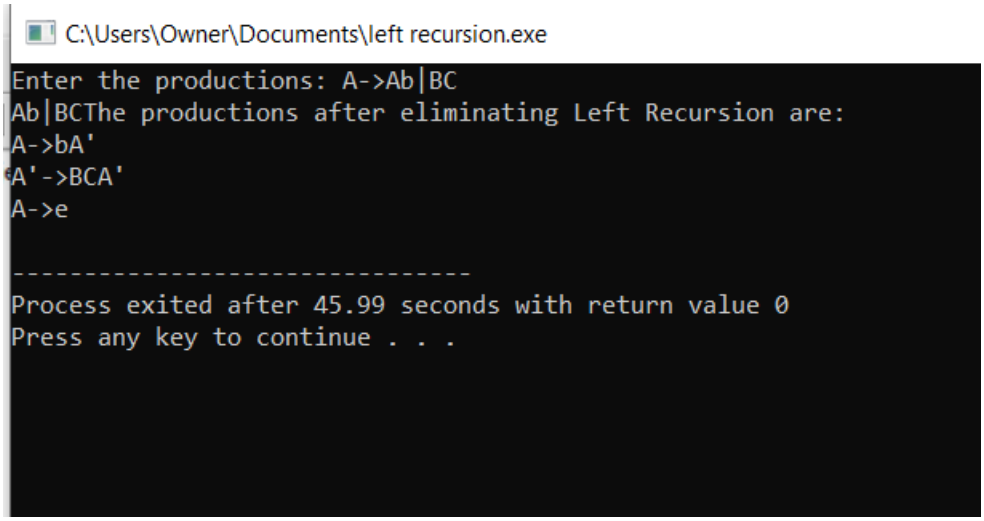
1.) left recursion :

```
#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);
}
```

```
C:\Users\Owner\Documents\left factoring.exe
Enter Production : A->BC|Bc

 A->BX
 X->C|c

--------------------------------
Process exited after 58.31 seconds with return value 0
Press any key to continue . . .
```

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);
    }
```

C:\Users\Owner\Documents\symbol table.exe

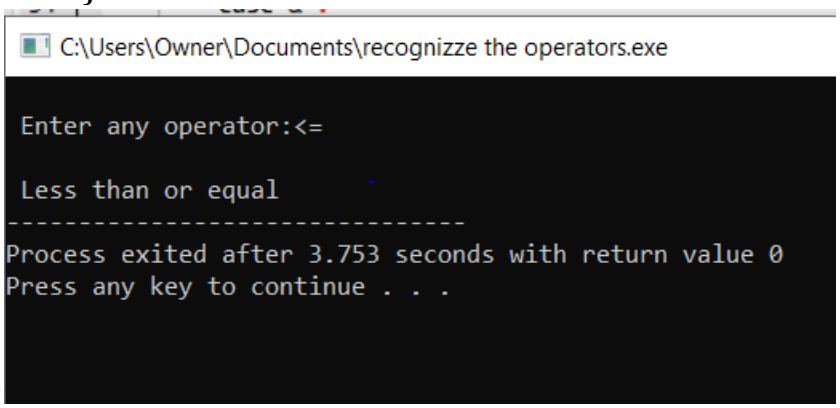
```
label is found
1.insert
2.display
3.search
4.modify
5.exit

4
enter the labe:label1
label is found
enter the address150

1.insert
2.display
3.search
4.modify
5.exit
2
label1  150

1.insert
2.display
3.search
4.modify
5.exit
5

--------------------------------
Process exited after 154.4 seconds with return value 0
Press any key to continue . . .
```

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");
    }
```



}

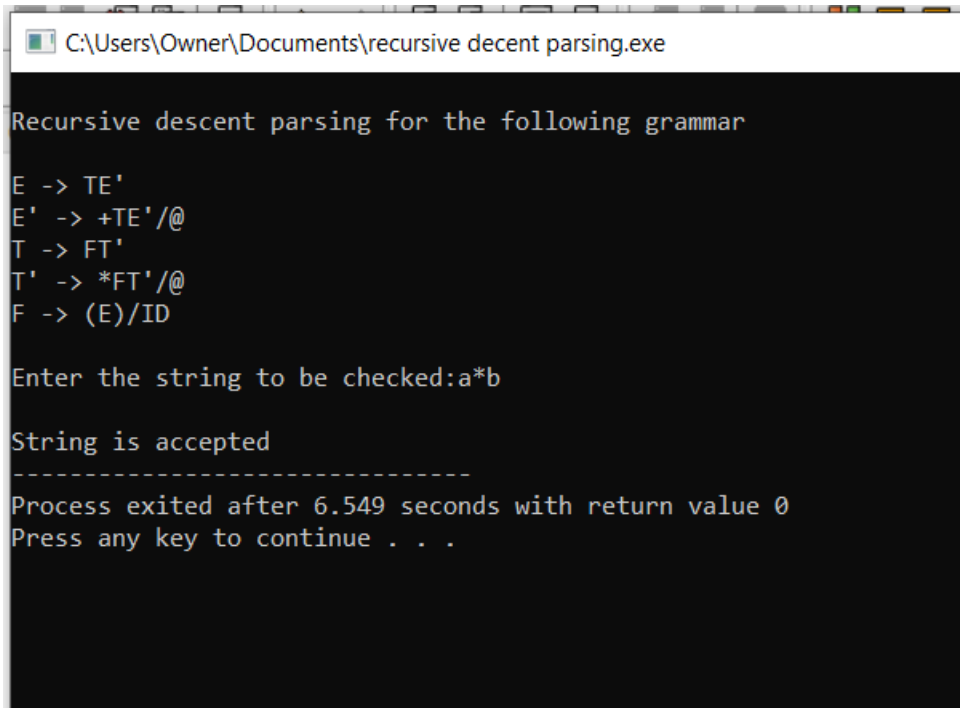
5.) recursive decent parsing:

```
#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");
    }
```

C:\Users\Owner\Documents\comments.exe

```
 Enter comment:This is a comment

 It is not a comment
--------------------------------
Process exited after 97.1 seconds with return value 0
Press any key to continue . . .
```

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]);
     }
     }
```

C:\Users\Owner\Documents\id,op.exe

```
enter the string : a = b + 3 - 5 * 2
 identifiers : a b
 constants : 3 5 2
 operators : = + - *
--------------------------------
Process exited after 37.44 seconds with return value 0
Press any key to continue . . .
```

8) shift reduce parsing:

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

#define MAX 100

char stack[MAX];
int top = -1;

char input[MAX];

void push(char c) {
    stack[++top] = c;
}

void pop() {
    if (top >= 0) {
        top--;
    }
}

void displayStack() {
    for (int i = 0; i <= top; i++) {
        printf("%c", stack[i]);
    }
    printf("\t");
}

void displayInput(int i) {
    printf("%s\n", input + i);
}


int checkReduction() {
    if (top >= 2) {
        if (stack[top] == 'E' && (stack[top-1] == '+' || stack[top-1] == '*') && stack[top-2] == 'E') {
            pop(); pop(); pop();
            push('E');
            return 1;
        }
    }

    if (top >= 2) {
        if (stack[top] == ')' && stack[top-1] == 'E' && stack[top-2] == '(') {
            pop(); pop(); pop();
            push('E');
            return 1;
        }
    }
```

```
    if (top >= 0) {
        if (stack[top] == 'i') {
            pop();
            push('E');
            return 1;
        }
    }

    return 0;
}

int main() {
    printf("Enter the input string:\n");
    fgets(input, MAX, stdin);
    input[strcspn(input, "\n")] = '\0';

    printf("STACK\tINPUT\n");
    printf("-----\t-----\n");

    int i = 0;
    char currentSymbol[3];

    while (input[i] != '\0') {
        if (input[i] != ' ') {
            currentSymbol[0] = input[i];
            if (input[i + 1] == 'd') {
                currentSymbol[1] = input[i + 1];
                currentSymbol[2] = '\0';
                push('i');
                i += 2;
            } else {
                push(input[i]);
                i++;
            }
        } else {
            i++;
        }

        displayStack();
        displayInput(i);

        while (checkReduction()) {
            displayStack();
            displayInput(i);
        }
    }

    while (checkReduction()) {
        displayStack();
        printf("\n");
    }
```

```
    if (top == 0 && stack[top] == 'E') {
        printf("\nString is accepted.\n");
    } else {
        printf("\nString is rejected.\n");
    }

    return 0;
}
```

```
Enter the input string:
id + id * id
STACK   INPUT
-----   -----
i        + id * id
E        + id * id
E       + id * id
E+       id * id
E+      id * id
E+i      * id
E+E      * id
E        * id
E       * id
E*       id
E*      id
E*i
E*E
E

String is accepted.
```

9.)lines chars and words:

```
#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;
}
```

C:\Users\Owner\Documents\lines char word.exe

```
Enter text (Ctrl+D to end):
i went to my village to meet my relatives
i went to my native place to meet my parents
^Z
Characters: 87
Words: 19
Lines: 3

--------------------------------
Process exited after 5.931 seconds with return value 0
Press any key to continue . . .
```

10.)three address

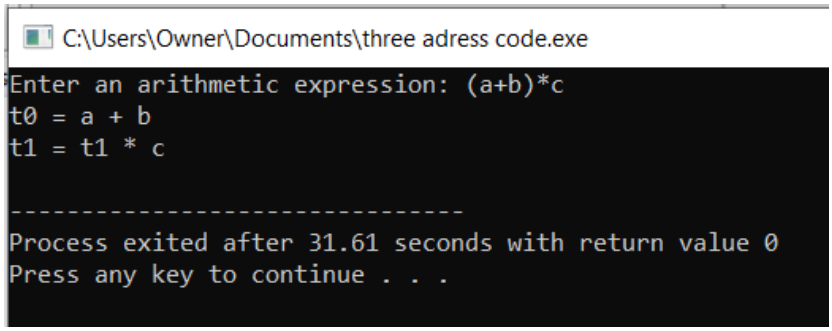
code :

```
 #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 variable
char* 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;
}
```



11) back end implementation:

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

#define MAX 100

int stack[MAX];
int top = -1;
int registerCount = 0;

void push(int value) {
    stack[++top] = value;
}

int pop() {
    return stack[top--];
}

void generateAssembly(char op, int operand1, int operand2) {
```

```
    printf("LOAD R%d, %d\n", registerCount, operand1);
    printf("%s R%d, %d\n", (op == '+') ? "ADD" :
                           (op == '-') ? "SUB" :
                           (op == '*') ? "MUL" : "DIV",
                           registerCount, operand2);
    push(registerCount);
    registerCount++;
}

void evaluatePostfix(char* expression) {
    int i = 0, operand1, operand2;

    printf("Generated Assembly Code:\n");

    while (expression[i] != '\0') {
        if (isdigit(expression[i])) {
            push(expression[i] - '0');
        } else {
            operand2 = pop();
            operand1 = pop();
            generateAssembly(expression[i], operand1, operand2);
        }
        i++;
    }

    printf("STORE RESULT, R%d\n", pop());
}

int main() {
    char postfix[MAX];

    printf("Enter a postfix expression: ");
    scanf("%s", postfix);

    evaluatePostfix(postfix);

    return 0;
}
```

```
Enter a postfix expression: 53+2*
Generated Assembly Code:
LOAD R0, 5
ADD R0, 3
LOAD R1, 0
MUL R1, 2
STORE RESULT, R1

--------------------------------
Process exited after 11.9 seconds with return value 0
Press any key to continue . . .
```

12)top down and bottom up parsing:

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

#define MAX 100

char stack[MAX];
int top = -1;
char input[MAX];

int i = 0;
int error = 0;

void push(char c) {
    stack[++top] = c;
}

void pop() {
    if (top >= 0) {
        top--;
    }
}

void displayStack() {
    for (int i = 0; i <= top; i++) {
        printf("%c", stack[i]);
    }
}

void E();
void Eprime();
void T();
```

```
void Tprime();
void F();

void E() {
    printf("E -> T E'\n");
    T();
    Eprime();
}

void Eprime() {
    if (input[i] == '+') {
        printf("E' -> + T E'\n");
        i++;
        T();
        Eprime();
    } else {
        printf("E' -> e\n");
    }
}

void T() {
    printf("T -> F T'\n");
    F();
    Tprime();
}

void Tprime() {
    if (input[i] == '*') {
        printf("T' -> * F T'\n");
        i++;
        F();
        Tprime();
    } else {
        printf("T' -> e\n");
    }
}

void F() {
    if (input[i] == '(') {
        printf("F -> ( E )\n");
        i++;
        E();
        if (input[i] == ')') {
            i++;
        } else {
            error = 1;
        }
    } else if (isalnum(input[i])) {
        printf("F -> id\n");
        i++;
    } else {
```

```
        error = 1;
    }
}

int reduce() {
    if (top >= 2) {
        if (stack[top] == 'E' && stack[top-1] == '+' && stack[top-2] == 'E') {
            pop(); pop(); pop(); push('E');
            return 1;
        }
    }

    if (top >= 2) {
        if (stack[top] == 'E' && stack[top-1] == '*' && stack[top-2] == 'E') {
            pop(); pop(); pop(); push('E');
            return 1;
        }
    }

    if (top >= 2) {
        if (stack[top] == ')' && stack[top-1] == 'E' && stack[top-2] == '(') {
            pop(); pop(); pop(); push('E');
            return 1;
        }
    }

    if (top >= 0) {
        if (stack[top] == 'i') {
            pop(); push('E');
            return 1;
        }
    }

    return 0;
}

int main() {
    int choice;

    printf("Enter the arithmetic expression: ");
    scanf("%s", input);

    printf("Choose Parsing Method:\n");
    printf("1. Top-Down Parsing (Recursive Descent)\n");
    printf("2. Bottom-Up Parsing (Shift-Reduce)\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    if (choice == 1) {
        printf("\nTop-Down Parsing (Recursive Descent):\n");
        E();
```

```
        if (strlen(input) == i && error == 0) {
            printf("String is accepted.\n");
        } else {
            printf("String is rejected.\n");
        }
    } else if (choice == 2) {
        printf("\nBottom-Up Parsing (Shift-Reduce):\n");
        int i = 0;
        printf("STACK\tINPUT\tACTION\n");
        printf("-----\t-----\t------\n");

        while (input[i] != '\0') {
            push(input[i]);
            i++;
            displayStack();
            printf("\t%s\tSHIFT\n", input + i);

            while (reduce()) {
                displayStack();
                printf("\t%s\tREDUCE\n", input + i);
            }
        }

        while (reduce()) {
            displayStack();
            printf("\t\tREDUCE\n");
        }

        if (top == 0 && stack[top] == 'E') {
            printf("String is accepted.\n");
        } else {
            printf("String is rejected.\n");
        }
    } else {
        printf("Invalid choice.\n");
    }

    return 0;
}
```

```
Enter the arithmetic expression: id+id*id
Choose Parsing Method:
1. Top-Down Parsing (Recursive Descent)
2. Bottom-Up Parsing (Shift-Reduce)
Enter your choice: 1

Top-Down Parsing (Recursive Descent):
E -> T E'
T -> F T'
F -> id
T' -> e
E' -> e
String is rejected.

--------------------------------
Process exited after 10.42 seconds with return value 0
Press any key to continue . . .
```

```
Enter the arithmetic expression: id+id*id
Choose Parsing Method:
1. Top-Down Parsing (Recursive Descent)
2. Bottom-Up Parsing (Shift-Reduce)
Enter your choice: 2

Bottom-Up Parsing (Shift-Reduce):
STACK   INPUT   ACTION
-----   -----   ------
i       d+id*id SHIFT
E       d+id*id REDUCE
Ed      +id*id  SHIFT
Ed+     id*id   SHIFT
Ed+i    d*id    SHIFT
Ed+E    d*id    REDUCE
Ed+Ed   *id     SHIFT
Ed+Ed*  id      SHIFT
Ed+Ed*i d       SHIFT
Ed+Ed*E d       REDUCE
Ed+Ed*Ed                SHIFT
String is rejected.

--------------------------------
Process exited after 13.88 seconds with return value 0
Press any key to continue . . .
```