**DAY 5**

**EXPERIMENT-33**: Implement a C program to perform symbol table operations.

#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()

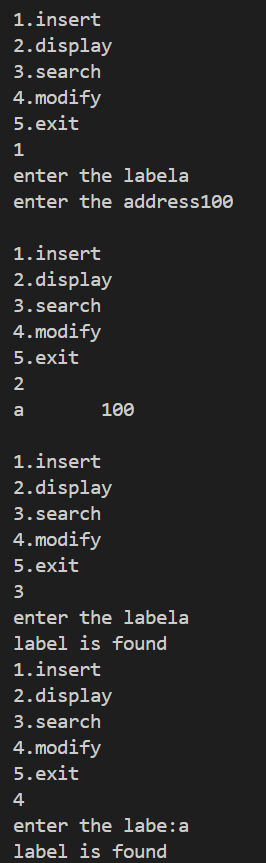
{

    int i;

    for(i=0;i<cnt;i++)

    printf("%s\t%d\n",sy[i].label,sy[i].addr);

}



**EXPERIMENT-34** implement to check whether the given input string is satisfying the grammar or not .

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main() {

    char string[50];

    int flag,count=0;

    printf("The grammar is: S->aS, S->Sb, S->ab\n");

    printf("Enter the string to be checked:\n");

    gets(string);

    if(string[0]=='a') {

        flag=0;

        for (count=1;string[count-1]!='\0';count++) {

            if(string[count]=='b') {

                flag=1;

                continue;

            } else if((flag==1)&&(string[count]=='a')) {

                printf("The string does not belong to the specified grammar");

                break;

            } else if(string[count]=='a')

            continue; else if((flag==1)&&(string[count]='\0')) {

                printf("String not accepted…..!!!!");

                break;

            } else {

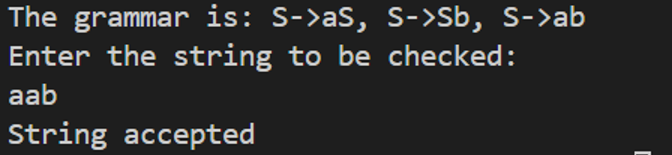
                printf("String accepted");

            }

        }

    }

}



**EXPERIMENT-35:** Write a C program to construct recursive descent parsing.

#include<stdio.h>

#include<conio.h>

#include<string.h>

char input[100];

int i,l;

void main()

{

//clrscr();

    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:");

    gets(input);

    if(E())

    {

        if(input[i+1]=='\0')

            printf("\nString is accepted");

        else

            printf("\nString is not accepted");

    }

    else

        printf("\nString not accepted");

        getch();

}

E()

{

    if(T())

    {

        if(EP())

            return(1);

        else

            return(0);

    }

    else

        return(0);

}

EP()

{

    if(input[i]=='+')

    {

        i++;

        if(T())

        {

            if(EP())

                return(1);

            else

                return(0);

        }

        else

            return(0);

    }

    else

        return(1);

}

T()

{

    if(F())

    {

        if(TP())

            return(1);

        else

            return(0);

    }

    else

        return(0);

}

TP()

{

    if(input[i]=='\*')

    {

        i++;

        if(F())

        {

            if(TP())

                return(1);

            else

                return(0);

        }

        else

            return(0);

    }

    else

        return(1);

}

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++;

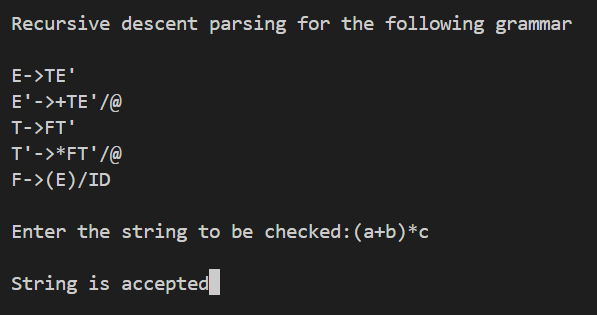
        return(1);

    }

    else

        return(0);

}



**EXPERIMENT-36:** C Program to help the students to understand about the operator precedence parsing for an expression

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX 100

// Define precedence levels

int precedence(char operator) {

    switch (operator) {

        case '+':

        case '-':

            return 1;

        case '\*':

        case '/':

            return 2;

        case '^':

            return 3;

        default:

            return 0;

    }

}

// Define associativity of operators

int is\_right\_associative(char operator) {

    if (operator == '^')

        return 1;

    return 0;

}

// Check if character is an operator

int is\_operator(char ch) {

    return (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^');

}

// Perform the actual operator precedence parsing

void operator\_precedence\_parsing(char \*expression) {

    char stack[MAX];

    int top = -1;

    char output[MAX];

    int output\_index = 0;

    for (int i = 0; i < strlen(expression); i++) {

        if (isdigit(expression[i])) {

            output[output\_index++] = expression[i];

        } else if (is\_operator(expression[i])) {

            while (top != -1 && precedence(stack[top]) >= precedence(expression[i]) && !is\_right\_associative(expression[i])) {

                output[output\_index++] = stack[top--];

            }

            stack[++top] = expression[i];

        }

    }

    while (top != -1) {

        output[output\_index++] = stack[top--];

    }

    output[output\_index] = '\0';

    printf("Parsed expression (Postfix notation): %s\n", output);

}

int main() {

    char expression[MAX];

    printf("Enter the expression: ");

    scanf("%s", expression);

    operator\_precedence\_parsing(expression);

    return 0;

}



**EXPERIMENT-37:** Write a C Program to Generate the Three address code representation for the given input statement.

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

struct three

{

char data[10],temp[7];

}s[30];

int main()

{

char d1[7],d2[7]="t";

int i=0,j=1,len=0;

FILE \*f1,\*f2;

//clrscr();

f1=fopen("sum.txt","r");

f2=fopen("res.txt","w");

while(fscanf(f1,"%s",s[len].data)!=EOF)

len++;

itoa(j,d1,7);

strcat(d2,d1);

strcpy(s[j].temp,d2);

strcpy(d1,"");

strcpy(d2,"t");

if(!strcmp(s[3].data,"+"))

{

fprintf(f2,"%s=%s+%s",s[j].temp,s[i+2].data,s[i+4].data);

j++;

}

else if(!strcmp(s[3].data,"-"))

{

fprintf(f2,"%s=%s-%s",s[j].temp,s[i+2].data,s[i+4].data);

j++;

}

for(i=4;i<len-2;i+=2)

{

itoa(j,d1,7);

strcat(d2,d1);

strcpy(s[j].temp,d2);

if(!strcmp(s[i+1].data,"+"))

fprintf(f2,"\n%s=%s+%s",s[j].temp,s[j-1].temp,s[i+2].data);

else if(!strcmp(s[i+1].data,"-"))

fprintf(f2,"\n%s=%s-%s",s[j].temp,s[j-1].temp,s[i+2].data);

strcpy(d1,"");

strcpy(d2,"t");

j++;

}

fprintf(f2,"\n%s=%s",s[0].data,s[j-1].temp);

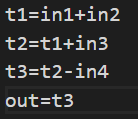
fclose(f1);

fclose(f2);

getch();

}





**EXPERIMENT-38:** Write a C program for implementing a Lexical Analyzer to Count the number of characters, words, and lines .

#include <stdio.h>

int main()

{

    char str[100];//input string with size 100

    int words=0,newline=0,characters=0; // counter variables

    scanf("%[^~]",&str);//scanf formatting

    for(int i=0;str[i]!='\0';i++)

     {

         if(str[i] == ' ')

         {

              words++;

         }

         else if(str[i] == '\n')

         {

             newline++;

              words++;//since with every next line new words start. corner case 1

         }

         else if(str[i] != ' ' && str[i] != '\n'){

         characters++;

         }

     }

    if(characters > 0)//Corner case 2,3.

    {

        words++;

        newline++;

    }

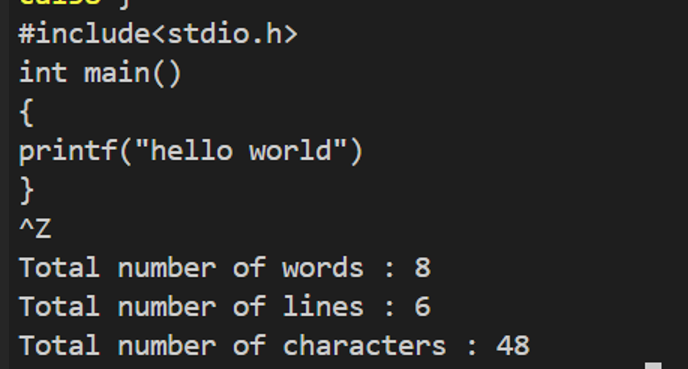
     printf("Total number of words : %d\n",words);

     printf("Total number of lines : %d\n",newline);

     printf("Total number of characters : %d\n",characters);

    return 0;

}



**EXPERIMENT-39:** Write a C Program for code optimization to eliminate common subexpression.

#include <stdio.h>

#include <string.h>

#define MAX 100

typedef struct {

    char left[MAX];

    char right[MAX];

} Expression;

void eliminate\_common\_subexpressions(Expression expressions[], int count) {

    int i, j, k;

    int optimized\_count = 0;

    Expression optimized[MAX];

    for (i = 0; i < count; i++) {

        int found = 0;

        for (j = 0; j < optimized\_count; j++) {

            if (strcmp(expressions[i].right, optimized[j].right) == 0) {

                printf("%s = %s\n", expressions[i].left, optimized[j].left);

                found = 1;

                break;

            }

        }

        if (!found) {

            optimized[optimized\_count++] = expressions[i];

            printf("%s = %s\n", expressions[i].left, expressions[i].right);

        }

    }

}

int main() {

    int count, i;

    Expression expressions[MAX];

    printf("Enter the number of expressions: ");

    scanf("%d", &count);

    getchar(); // Consume the newline character

    printf("Enter the expressions (e.g., a = b + c):\n");

    for (i = 0; i < count; i++) {

        printf("Expression %d: ", i + 1);

        fgets(expressions[i].left, MAX, stdin);

        // Split the input into left and right parts

        char \*token = strtok(expressions[i].left, "=");

        strcpy(expressions[i].left, token);

        token = strtok(NULL, "\n");

        strcpy(expressions[i].right, token);

        // Remove any leading/trailing spaces

        expressions[i].left[strcspn(expressions[i].left, "\n")] = '\0';

        expressions[i].right[strcspn(expressions[i].right, "\n")] = '\0';

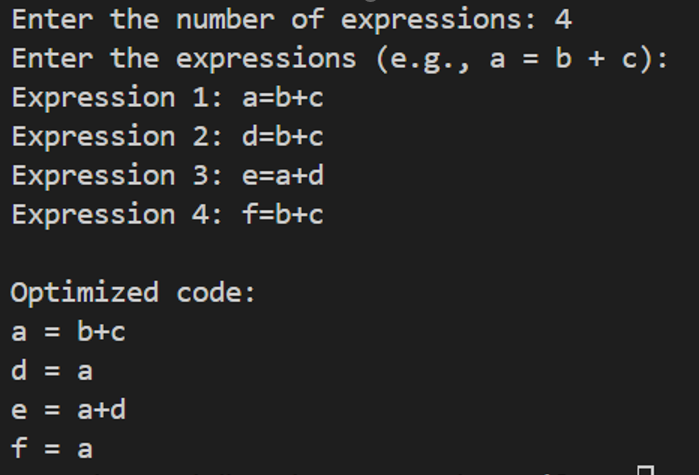
    }

    printf("\nOptimized code:\n");

    eliminate\_common\_subexpressions(expressions, count);

    return 0;

}



**EXPERIMENT-40:** Write a C program to implement the back end of the compiler.

#include<stdio.h>

#include<conio.h>

#include<string.h>

int main()

{

    int n,i,j;

    char a[50][50];

    printf("enter the no: intermediate code:");

    scanf("%d",&n);

    for(i=0;i<n;i++)

    {

        printf("enter the 3 address code:%d:",i+1);

        for(j=0;j<6;j++)

        {

            scanf("%c",&a[i][j]);

        }

    }

    printf("the generated code is:");

    for(i=0;i<n;i++)

    {

        printf("\n mov %c,R%d",a[i][3],i);

        if(a[i][4]=='-')

        {

            printf("\n sub %c,R%d",a[i][5],i);

        }

        if(a[i][4]=='+')

        {

            printf("\n add %c,R%d",a[i][5],i);

        }

        if(a[i][4]=='\*')

        {

            printf("\n mul %c,R%d",a[i][5],i);

        }

        if(a[i][4]=='/')

        {

            printf("\n div %c,R%d",a[i][5],i);

        }

        printf("\n mov R%d,%c",i,a[i][1]);

        printf("\n");

    }

    return 0;

}

