EXPERIMENT NO 1
LEXICAL ANALYSER

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 07/08/2024

CODE:

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

int isKeyword(char buffer[])
{
	char keywords[35][10] =
	{"auto", "break", "case", "char", "const", "continue", "default", "do", "double", "else",
"enum", "extern",
	"float", "for", "goto","if", "int", "long", "register", "return", "short", "signed", "sizeof",
"static", "struct",
	"switch", "typdef", "union", "unsigned", "void", "volatile", "while", "printf",
"scanf","main"};

	int i,flag =0;
	for(i=0;i<35;i++)
	{
		if(strcmp(keywords[i],buffer)==0)
		{
			flag=1;
			break;
		}
	}
	return flag;
}

void main()
{
	char ch,ch1, buffer[50], operator[]="+-*/%";
	FILE *fp,*fp2;
	int i, j=0, l=1, index = 1;
	char arr[1000][3];
	fp=fopen("inp.txt", "r");
	fp2=fopen("output.txt", "w+");
	if(fp==NULL)
	{
		printf("Error while opening file\n");
		exit(0);
	}
	else
	{
		printf("Lexeme: \t L.no:\t Token:");
		fprintf(fp2,"Lexeme: \t L.no:\t Token:");
		while((ch=fgetc(fp))!=EOF)
		{
			for(i=0;i<5;++i)
			{
				if(ch==operator[i])
				{
					printf("\n%c \t\t %d\t Arithmetic Operator ", ch,l);
					break;
				}
			}
			if(i==5)
			{
				if(isalnum(ch))
				{
					if(isdigit(ch) && j==0)
					{
						j=0;
					}
					else
					{
						buffer[j++] = ch;
						ch1= fgetc(fp);
						if(isalnum(ch1))
							fseek(fp, -1, SEEK_CUR);
						else
						{
							fseek (fp,-1,SEEK_CUR);
							buffer[j]='\0';
							if(isKeyword(buffer)==1)
							{
								if(strcmp(buffer, "printf")==0){
								while((ch=fgetc(fp))!='\n'){

								}
								}
								printf("\n%s \t\t %d \t keyword ", buffer,
l);

								j=0;
							}
							else
							{
								if(strcmp(buffer, "main")!=0)

								{
									fseek(fp2, SEEK_SET, 0);
									char a[50], b[50], c[50], ch2, ch3;
									int flag1 = 0;
```

```
                                    while(!feof(fp2)){
                                    fscanf(fp2, "%s\t%s\t%s", a, b,
c);
                                    if(strcmp(a, buffer) == 0){
                                    flag1 = 1;
                                    break;
                                }
                            }
                                if(flag1==0){
                                    ch2 = fgetc(fp);
                                    if(ch2=='='){
                                    ch3 = fgetc(fp);
                                    fprintf(fp2,"\n%s \t\t %d\t
identifier, %c", buffer, index, ch3);
                                    fseek(fp,-2,SEEK_CUR);
                                    }
                                    else{
                                    fprintf(fp2,"\n%s \t\t %d\t
identifier", buffer, index);
                                    }
                                    printf("\n%s \t\t %d\t Identifier,
%d", buffer, l, index);
                                    index++;
                                    }
                                else{
                                    printf("\n%s \t\t %d\t Identifier,
%s", buffer, l, b);
                                    }
                                    }
                                j=0;
                            }
                        }
                    }
                }
                if(ch=='<')
                {
                    ch1= fgetc(fp);
                    if(ch1=='=')
                    {
                        printf("\n%c%c \t\t %d \t Relop LE",ch,ch1,l);
                    }
                    else
                    {
                        fseek(fp,-1,SEEK_CUR);
                        printf("\n%c \t\t %d \t RelOP LT", ch,l);
                    }
                }
                else if(ch=='>')
                {
                    ch1= fgetc(fp);
                    if(ch1=='=')
                    {
```

```
                                        printf("\n%c%c \t\t %d \t Relop GE",ch,ch1,l);
                                }
                                else
                                {
                                        fseek(fp,-1,SEEK_CUR);
                                        printf("\n%c \t\t %d \t Relop GT", ch,l);
                                }
                        }
                        else if(ch=='=')
                        {
                                ch1=fgetc(fp);
                                if(ch1=='=')
                                {
                                        printf("\n%c%c \t\t %d \t Relop EQ", ch, ch1, l);
                                }
                                else
                                {
                                        fseek(fp, -1, SEEK_CUR);
                                        printf("\n%c \t\t %d \t Assign OP, EQ",ch,l);
                                }
                        }
                        if(ch=='\n'){
                                l++;
                        }
                }
        }
    }
    printf("\n");
}
```

INPUT.TXT:

```
void main(){

    int a,b,c=0;
    char ch;
    a=b+c;
    if(a<=b)
        a=b;
    printf("Hello World");
}
```

OUTPUT:

```
Lexeme:         L.no:   Token:
void            1       keyword
main            1       keyword
int             3       keyword
a               3       Identifier, 1
b               3       Identifier, 2
c               3       Identifier, 3
=               3       Assign OP, EQ
```

```
char        4    keyword
ch          4    Identifier, 4
a           5    Identifier, 1
=           5    Assign OP, EQ
b           5    Identifier, 2
+           5    Arithmetic Operator
c           5    Identifier, 3
if          6    keyword
a           6    Identifier, 1
<=          6    Relop LE
b           6    Identifier, 2
a           7    Identifier, 1
=           7    Assign OP, EQ
b           7    Identifier, 2
printf      8    keyword
```

OUTPUT.TXT:

```
Lexeme:     L.no:  Token:
a           1    identifier
b           2    identifier
c           3    identifier, 0
ch          4    identifier
```

EXPERIMENT NO 2
OPERATOR PRECEDENCE PARSER

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 07/08/2024

CODE:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
char *input;
int i=0;
char lasthandle[6],stack[50],handles[][7]={")E(","E*E","E+E","i","E^E","E-E","E/E"};
//(E) becomes )E( when pushed to stack

int top=0,l;
char prec[9][9]={

                        /*input*/

          /*stack    +    -   *   /   ^   i   (   )   $  */

          /*  + */  '>', '>','<','<','<','<','<','>','>',

          /*  - */  '>', '>','<','<','<','<','<','>','>',

          /*  * */  '>', '>','>','>','<','<','<','>','>',

          /*  / */  '>', '>','>','>','<','<','<','>','>',

          /*  ^ */  '>', '>','>','>','<','<','<','>','>',

          /*  i */  '>', '>','>','>','>','e','e','>','>',

          /*  ( */  '<', '<','<','<','<','<','<','>','e',

          /*  ) */  '>', '>','>','>','>','e','e','>','>',

          /*  $ */  '<', '<','<','<','<','<','<','<','>',

              };

int getindex(char c)
{
switch(c)
```

```
    {
    case '+':return 0;
    case '-':return 1;
    case '*':return 2;
    case '/':return 3;
    case '^':return 4;
    case 'i':return 5;
    case '(':return 6;
    case ')':return 7;
    case '$':return 8;
    }
}


int shift()
{
stack[++top]=*(input+i++);
stack[top+1]='\0';
}


int reduce()
{
int i,len,found,t;
for(i=0;i<7;i++)//selecting handles
    {
    len=strlen(handles[i]);
    if(stack[top]==handles[i][0]&&top+1>=len)
        {
        found=1;
        for(t=0;t<len;t++)
            {
            if(stack[top-t]!=handles[i][t])
                {
                found=0;
                break;
                }
            }
        if(found==1)
            {
            stack[top-t+1]='E';
            top=top-t+1;
            strcpy(lasthandle,handles[i]);
            stack[top+1]='\0';
            return 1;//successful reduction
            }
        }
   }
return 0;
}
```

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



void dispinput()
{
int j;
for(j=i;j<l;j++)
   printf("%c",*(input+j));
}



void main()
{
int j;

input=(char*)malloc(50*sizeof(char));
printf("\nEnter the string\n");
scanf("%s",input);
input=strcat(input,"$");
l=strlen(input);
strcpy(stack,"$");
printf("\nSTACK\tINPUT\tACTION");
while(i<=l)
         {
         shift();
         printf("\n");
         dispstack();
         printf("\t");
         dispinput();
         printf("\tShift");
         if(prec[getindex(stack[top])][getindex(input[i])]=='>')
                  {
                  while(reduce())
                           {
                           printf("\n");
                           dispstack();
                           printf("\t");
                           dispinput();
                           printf("\tReduced: E->%s",lasthandle);
                           }
                  }
         }

if(strcmp(stack,"$E$")==0)
```

```
    printf("\nAccepted;");
else
    printf("\nNot Accepted;");
}
```

OUTPUT

```
Enter the string
i*(i+i)

Enter the string
i*(i+i)

Enter the string
i*(i+i)

STACK        INPUT        ACTION
$i           *(i+i)$      Shift
$E           *(i+i)$      Reduced: E->i
$E*          (i+i)$       Shift
$E*(         i+i)$        Shift
$E*(i        +i)$         Shift
$E*(E        +i)$         Reduced: E->i
$E*(E+       i)$          Shift
$E*(E+i      )$           Shift
$E*(E+E      )$           Reduced: E->i
$E*(E        )$           Reduced: E->E+E
$E*(E)       $            Shift
$E*E         $            Reduced: E->)E(
$E           $            Reduced: E->E*E
$E$                       Shift
$E$                       Shift
Accepted;
```

EXPERIMENT NO 3
RECURSIVE DESCENT PARSER

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 14/08/2024

CODE:

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

char input[100];
int i, err;

void E();
void Eprime();
void T();
void Tprime();
void F();

void E() {
    T();
    Eprime();
}

void Eprime() {
    if (input[i] == '+') {
        i++;
        T();
        Eprime();
    } else if (input[i] == '-') {
        i++;
        T();
        Eprime();
    }
}

void T() {
    F();
    Tprime();
}

void Tprime() {
    if (input[i] == '*') {
        i++;
        F();
        Tprime();
    } else if (input[i] == '/') {
        i++;
        F();
        Tprime();
    }
}

void F() {
    if (isalnum(input[i])) {
        i++;
    } else if (input[i] == '(') {
        i++;
        E();
        if (input[i] == ')') {
            i++;
        } else {
            err = 1;
        }
    } else {
        err = 1;
    }
}

int main() {
    i = 0;
    err = 0;
    printf("Enter an expression: ");
    fgets(input, sizeof(input), stdin);

    // Remove newline character if present
    input[strcspn(input, "\n")] = '\0';

    E();
    if (strlen(input) == i && err == 0) {
        printf("%s is accepted\n", input);
    } else {
        printf("%s is rejected\n", input);
    }

    return 0;
}
```

OUTPUT

```
Enter an expression: (i+i)*i
(i+i)*i is accepted
```

EXPERIMENT NO 4
FIRST AND FOLLOW

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 14/08/2024

CODE:

```
#include<stdio.h>
#include<ctype.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,p1 = 0, p2, tmp;
char firstmat[10][100],followmat[10][100], production[10][10],f[10], first[10];

int main(int argc, char ** argv) {
        char ck;
        int jm = 0;
        int km = 0;
        int i, choice;
        char c, ch;
        printf("\nEnter the productions: ");

        scanf("%d", & count);
        for (i = 0; i < count; i++)
                scanf("%s", production[i]);
        int kay;
        char done[count];
        int ptr = -1;
        for (k = 0; k < count; k++) {
                for (kay = 0; kay < 100; kay++)
                        firstmat[k][kay] = '!';
        }
        int p1 = 0, p2, tmp;
        for (k = 0; k < count; k++) {
                c = production[k][0];
                p2 = 0;
                tmp = 0;
                for (kay = 0; kay <= ptr; kay++)
                if (c == done[kay])
```

```
		tmp = 1;
		if (tmp == 1)
		continue;
		findfirst(c, 0, 0);
		ptr += 1;
		done[ptr] = c;
		printf("\nFirst(%c) = { ", c);
		firstmat[p1][p2++] = c;
		for (i = 0 + jm; i < n; i++) {
			int lark = 0, chk = 0;
			for (lark = 0; lark < p2; lark++) {
				if (first[i] == firstmat[p1][lark]) {
					chk = 1;
					break;
				}
			}
			if (chk == 0) {
				printf("%c, ", first[i]);
				firstmat[p1][p2++] = first[i];
			}
		}
		printf("}\n");
		jm = n;
		p1++;
	}
	printf("\n\n\n");
	char donee[count];
	ptr = -1;
	for (k = 0; k < count; k++) {
		for (kay = 0; kay < 100; kay++)
			followmat[k][kay] = '!';
	}
	p1 = 0; //in p
	int kk, u, flag, land = 0;
	char s[10];
	u = 0;
	flag = 0;
	for (e = 0; e < count; e++) {
		kk = 0;
		ck = production[e][0];
		for (i = 0; i <= u; i++) {
			if (s[i] == ck) {
				kk++;
			}
		}
		if (kk == 0) {

			s[u] = ck;
			u++;
			p2 = 0;
			tmp = 0;
			for (kay = 0; kay <= ptr; kay++)
```

```
                }
                        if (ck == donee[kay])
                          tmp = 1;

                        if (tmp == 1)
                        continue;
                        land += 1;
                        follow(ck);

                        ptr += 1;
                        printf("Follow(%c) = { ", ck);
                        followmat[p1][p2++] = ck;
                        for (i = 0 + km; i < m; i++) {
                                int lark = 0, chk = 0;
                                for (lark = 0; lark < p2; lark++) {
                                          if (f[i] == followmat[p1][lark]) {
                                                chk = 1;
                                                break;
                                          }
                                }
                                if (chk == 0) {
                                        printf("%c, ", f[i]);
                                        followmat[p1][p2++] = f[i];
                                }
                        }
                        printf("}\n\n");
                        km = m;
                        p1++;
                }
        }
}

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] == production[j][2])
                        continue;

                if (production[j][0] == c) {
                        if (production[j][2] == '#')
                        {
                                if (production[q1][q2] == '\0')
                                  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 (firstmat[i][0] == c)
                        break;
                }
                while (firstmat[i][j] != '!'){
                        if (firstmat[i][j] != '#')
                        f[m++] = firstmat[i][j];
                        else {
                                if (production[c1][c2] == '\0')
                                  follow(production[c1][0]);
                                else
                                        followfirst(production[c1][c2], c1, c2 + 1);
                        }
                        j++;
                }
        }
}
```

OUTPUT

```
Enter the productions: 3
E=E+T
E=T
T=l

First(E) = { l, }

First(T) = { l, }



Follow(E) = { $, +, }

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

EXPERIMENT NO 5
INTERMEDIATE CODE GENERATION

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 21/08/2024

CODE:

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

char s[100], post[100], stack[100], queue[100];
int top = -1, j = 0, front = -1;

int precedence(char c){
	if(c == '+' || c == '-'){
		return 1;
	}
	else if(c == '*' || c == '/'){
		return 2;
	}
	else if(c == '^'){
		return 3;
	}
	else{
		return 0;
	}
}
```

```
void postfix(){
        int n = strlen(s), i = 0;
        while(s[i] != '\0'){
                if(s[i] == '('){
                        top++;
                        stack[top] = s[i];
                }
                else if(isalpha(s[i])){
                        post[j++] = s[i];
                }
                else if(precedence(s[i])){
                        while(precedence(stack[top]) >= precedence(s[i])){
                                post[j++] = stack[top--];
                        }
                        top++;
                        stack[top] = s[i];
                }
                else if(s[i] == ')'){
                        while(stack[top] != '('){
                                post[j++] = stack[top--];
                        }
                        top--;
                }
                i++;
        }
        while(top!=-1){
                post[j++] = stack[top--];
        }
        post[j] = '\0';
}

void main(){
        int i = 0;
        char ind = '1';

        FILE *fp,*fp1,*fp2,*fp3;
        fp = fopen( "3addr.txt","w");
        fp1 = fopen( "quadraple.txt","w");
        fp2 = fopen( "triple.txt","w");
        fprintf(fp,"Three Address\n");
        fprintf(fp1,"Quadruple\nOP\tO1\tO2\tRES\n");
        fprintf(fp2,"Triple\nIN\tOP\tO1\tO2\n");
        fp3=fopen( "input.txt","r");
        printf("Input the string: ");
        while((fscanf(fp3,"%s",s))!=EOF)
        {
        postfix();

        printf("Infix : %s \nPostfix : %s\n", s, post);
        while(post[i] != '\0'){
                if(precedence(post[i])){
                        char a = queue[front--];
                char b = queue[front--];
                if(isdigit(a) && isdigit(b)){
                    fprintf(fp,"t%c = t%c %c t%c\n", ind, b, post[i], a);
                    fprintf(fp1,"%c\tt%c\tt%c\tt%c\n", post[i], b, a, ind);
                }
                else if(isdigit(b)){
                    fprintf(fp,"t%c = t%c %c %c\n", ind, b, post[i], a);
                    fprintf(fp1,"%c\tt%c\t%c\tt%c\n", post[i], b, a, ind);
                }
                else if(isdigit(a)){
                    fprintf(fp,"t%c = %c %c t%c\n", ind, b, post[i], a);
                    fprintf(fp1,"%c\t%c\tt%c\tt%c\n", post[i], b, a, ind);
                }
                else{
                    fprintf(fp,"t%c = %c %c %c\n", ind, b, post[i], a);
                    fprintf(fp1,"%c\t%c\t%c\tt%c\n", post[i], b, a, ind);
                }
                fprintf(fp2,"%c\t%c\t%c\t%c\n", ind, post[i], b, a);
                front++;
                queue[front] = ind;
                ind++;
            }
            else{
                front++;
                queue[front] = post[i];
            }
            i++;
        }
        }
}
```

INPUT

```
a+b*c^d+e
a*b+c*d
```

OUTPUT

```
Input the string: Infix : a+b*c^d+e
Postfix : abcd^*+e+
Infix : a*b+c*d
Postfix : ab*cd*+
```

TRIPLE.TXT

```
Triple
IN      OP      O1      O2
1       ^       c       d
2       *       b       1
3       +       a       2
4       +       3       e
5       *       a       b
```

```
6       *       c       d
7       +       5       6
```

QUADRUPLE.TXT

```
Quadruple
OP      O1      O2      RES
^       c       d       t1
*       b       t1      t2
+       a       t2      t3
+       t3      e       t4
*       a       b       t5
*       c       d       t6
+       t5      t6      t7
```

THREEADRRESS.TXT

```
Three Address
t1 = c ^ d
t2 = b * t1
t3 = a + t2
t4 = t3 + e
t5 = a * b
t6 = c * d
t7 = t5 + t6
```

EXPERIMENT NO 6
CONSTANT PROPAGATION

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 04/09/2024

CODE:

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


char buf[30];
void showError(int ln,char buffer[], char c, int code){
printf("Error at Line Number %d in the lexeme %s%c\nTerminating Program! Error Code: %d\
n",ln,buffer,c,code);
exit(0);
}

char* toString(int a)
{
int l=0;
while(a>0)
{
	char c='0'+(a%10);
	a=a/10;
	buf[l]=c;
	l++;
}

for(int i=0;i<l/2;i++)
{
	char c=buf[i];
	buf[i]=buf[l-1-i];
	buf[l-1-i]=c;
}
}



int main()
{

char symbols[100][2][20]={

};

int symbc=0;
char keywords[40][20]={
	"void","int","char","exit","for","while","return","if","else","main","printf"
};
char dtypes[40][20]={"int","char","float"};


FILE *fp,*symtab,*fp2,*fp3;
fp = fopen( "inp.txt","r");
fp2=fopen("con.txt","w");
fp3 = fopen( "con.txt","r");
symtab = fopen( "result.txt","w");
if(fp==NULL)
{
	printf("File could not be opened!\n");
	exit(0);
}
if(symtab==NULL)
{
	printf("Result File could not be opened!\n");
	exit(0);
}
char buffer[30],idf[30],idf2[30];
char idstack[30][20],opstack[30];
int idtop=0,optop=0;
int ln=1;
bool isd=false;
bool expecting=false;
bool isStringLiteral=false;
bool defining=false;
int sp=0,cp=0,bp=0;
char c=fgetc(fp);
while(!feof(fp))
```

```
{
    if(isStringLiteral && c!='"')
    {

        fprintf(symtab,"%c",c);
        c=fgetc(fp);
        continue;
    }
    else if(!isStringLiteral && !isalnum(c)&&c!='_'&&bp!=0)
    {
        buffer[bp]='\0';
        bool isk=false;
        for(int i=0;i<40;i++)
        {
            if(strcmp(keywords[i],buffer)==0)
            {
                isk=true;
                break;
            }
        }
        for(int i=0;i<40;i++)
        {
            if(strcmp(dtypes[i],buffer)==0)
            {
                defining=true;
                break;
            }
        }
        if(!isk&&isd)
        {
            strcpy(idstack[idtop],buffer);
            idtop++;
        }
        else if(!isk)
        {
            strcpy(idstack[idtop],buffer);
            idtop++;
            strcpy(idf,buffer);
            bool flag=false;
            if(!expecting)
            {
            if(c!='=')
            {for(int i=0;i<symbc;i++)
                    {
                        if(strcmp(symbols[i][0],idf)==0)
                        {
                            fprintf(symtab,"%s",symbols[i][1]);
                            flag=true;
                            break;
                        }
                    }}
                    if(flag==false)
                fprintf(symtab,"%s",buffer);
            }
        }
        else
        {

              fprintf(symtab,"%s",buffer);
        }
        bp=0;
        buffer[bp]='\0';
        isd=false;
    }
    if(c=='"')
    {
        if(isStringLiteral)
            {
        buffer[bp]='\0';

            }
        isStringLiteral=!isStringLiteral;

        fprintf(symtab,"%c",c);
        c=fgetc(fp);
        continue;
    }
    else if(c=='+'||c=='-'||c=='*'||c=='/')
    {

        if(optop>0)
        {
            if((opstack[optop-1]=='*'||opstack[optop-1]=='/')&&(c=='+'||c=='-'))
            {
                  if(idtop<2)
                   {
                        showError(ln,buffer,c,1);
                   }
                   idtop--;
                   char operand[30];
                   strcpy(operand,idstack[idtop]);

                   float op1=0,op2=0;
                   if(isdigit(operand[0]))
                   {
                        op1=atof(operand);
                   }
                   else
                   {
                        int i=0;
                        for(i=0;i<symbc;i++)
                        {
                            if(strcmp(symbols[i][0],operand)==0)
                            {
```

```
                                        op1=atof(symbols[i][1]);
                                        break;
                                }
                        }
                        if(i==symbc)
                        {
                                showError(ln,buffer,c,2);
                        }
                }

                idtop--;
                strcpy(operand,idstack[idtop]);

                if(isdigit(operand[0]))
                {
                        op2=atof(operand);
                }
                else
                {
                        int i=0;
                        for(i=0;i<symbc;i++)
                        {
                                if(strcmp(symbols[i][0],operand)==0)
                                {
                                        op2=atof(symbols[i][1]);
                                        break;
                                }
                        }
                        if(i==symbc)
                        {
                                showError(ln,buffer,c,3);
                        }
                }
                float ans=0;
                if(opstack[optop-1]=='*')
                {
                        ans=op1*op2;
                }
                else if(opstack[optop-1]=='/')
                {
                        ans=op2/op1;
                }

                fp2=fopen("con.txt","w");
                fprintf(fp2,"%f\n",ans);
                fclose(fp2);
                fgets(buf,30,fp3);

                strcpy(idstack[idtop],buf) ;
                idtop++;
                optop--;
        }
```

```
		}
		opstack[optop]=c;
		optop++;
	}
	else if(isalpha(c))
	{
		if(isd)
		{
			showError(ln,buffer,c,4);
		}
		buffer[bp]=c;
		bp++;
		if(bp==1)
		{
			isd=false;
		}
	}
	else if(isdigit(c))
	{

		buffer[bp]=c;
		bp++;
		if(bp==1)
		{
			isd=true;
		}
	}
	else if(c=='_')
	{
		if(bp!=0)
		{
			buffer[bp]=c;
			bp++;
		}
		else
		{
			showError(ln,buffer,c,5);
		}
	}
	else if(c=='<'||c=='>'||c=='='||c=='!'){

		fprintf(symtab,"%c",c);
		char c2=fgetc(fp);
		if(c2!='='&&c=='!')
		{
			showError(ln,buffer,c,6);
			c=c2;
			continue;
		}
		else if(c!='=')
		{
			c=c2;
```

```
				expecting =true;
				continue;
			}
			else
			{
				c=c2;
				idtop=0;

				strcpy(idf2,idf);
				expecting=true;
				continue;
			}
		}
		else if(c==';'||c==',')
		{
			if(expecting)
			{
			while(idtop>1)
			{

					idtop--;
					char operand[30];

					strcpy(operand,idstack[idtop]);

					float op1=0,op2=0;
					if(isdigit(operand[0]))
					{
						op1=atoi(operand);
					}
					else
					{
						int i=0;
						for(i=0;i<symbc;i++)
						{
							if(strcmp(symbols[i][0],operand)==0)
							{
								if(strcmp(symbols[i][1],"-")==0)
									op1=0;
								else
									op1=atof(symbols[i][1]);
								break;
							}
						}
						if(i==symbc)
						{
							showError(ln,buffer,c,7);
						}
					}
					idtop--;
					strcpy(operand,idstack[idtop]);
			if(isdigit(operand[0]))
			{
				op2=atoi(operand);
			}
			else
			{
				int i=0;
				for(i=0;i<symbc;i++)
				{
					if(strcmp(symbols[i][0],operand)==0)
					{	if(strcmp(symbols[i][1],"-")==0)
							op1=0;
						else
						op2=atof(symbols[i][1]);
						break;
					}
				}
				if(i==symbc)
				{
					showError(ln,buffer,c,8);
				}
			}
			float ans=0;
			if(opstack[optop-1]=='*')
			{
				ans=op1*op2;
			}
			else if(opstack[optop-1]=='/')
			{
				ans=op2/op1;
			}
			else if(opstack[optop-1]=='+')
			{
				ans=op1+op2;
			}
			else if(opstack[optop-1]=='-')
			{
				ans=op2-op1;
			}
			fp2=fopen("con.txt","w");
			fprintf(fp2,"%f",ans);
			fclose(fp2);
			fp3 = fopen( "con.txt","r");
			fgets(buf,30,fp3);
			fclose(fp3);

			strcpy(idstack[idtop],buf) ;
			idtop++;
			optop--;
}
int targetInd=symbc;
if(!defining)
```

```
{
int j=0;
        for(j=0;j<symbc;j++)
                        {
                        if(strcmp(symbols[j][0],idf2)==0)
                        {
                        targetInd=j;
                        break;
                        }
                        }
}
strcpy(symbols[targetInd][0],idf2);
if(isalpha(idstack[0][0]))
{


        int i=0;
                        if(strcmp(symbols[targetInd][0],idf2)==0)
                        {
                        for(i=0;i<symbc;i++)
                        {


                                if(strcmp(symbols[i][0],idstack[0])==0)
                                {

                                        strcpy(symbols[targetInd][1],symbols[i][1]);
                                        break;
                                }

                        }
                        if(i==symbc)
                        {
                                showError(ln,buffer,c,12);
                        }
                        }


}
else
{
        int j=0;

        strcpy(symbols[targetInd][1],idstack[0]);



}


if(defining)
{symbc++;
}
```

```
                if(c==';')
                defining=false;

                fprintf(symtab,"%s",symbols[targetInd][1]);
                }
                else if(defining)
                {
                strcpy(symbols[symbc][0],idf);
                strcpy(symbols[symbc][1],"0");
                symbc++;
                if(c==';')
                defining=false;
                }
                expecting=false;

                fprintf(symtab,"%c",c);
        }
        else if(c=='{'||c=='}'||c=='('||c==')'||c=='['||c==']')
        {
                fprintf(symtab,"%c",c);
        }
        else if(c=='\n')
        {
                fprintf(symtab,"%c",c);
        }

        else if(c!=' '&&c!='\t')

        {
                showError(ln,buffer,c,10);
        }
        else
        {

                fprintf(symtab,"%c",c);
        }
        c=fgetc(fp);
}

fclose(symtab);
fclose(fp);
return 0;
}
```

INPUT.TXT

```
float pi=20/5;
void main()
{
        float c;
        c=2*pi;
}
```

CONS.TXT

```
8.000000
```

RESULT.TXT

```
float pi=4.000000;
void main()
{
        float c;
        c=8.000000;
}
```

EXPERIMENT NO 7
CODE OPTIMIZATION

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 11/09/2024

CODE:

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

#define MAX_QUADS 100

// Define structure to hold quadruples
typedef struct {
    char op[3];  // Operator
    char o1[5];  // Operand 1
    char o2[5];  // Operand 2
    char res[5]; // Result
} Quadruple;

// Function to check if a common subexpression exists
int is_common_subexpression(Quadruple q[], int index, char res[]) {
    for (int i = 0; i < index; i++) {
        // Check if the current operation and operands match any previous quadruple
        if (strcmp(q[i].op, q[index].op) == 0 && strcmp(q[i].o1, q[index].o1) == 0 && strcmp(q[i].o2,
q[index].o2) == 0) {
            strcpy(res, q[i].res);  // Copy the result of the common subexpression
            return 1;  // Found a common subexpression
        }
    }
    return 0;  // No common subexpression found
}

// Function to replace all occurrences of a redundant result in subsequent quadruples
void replace_redundant_result(Quadruple q[], int n, const char *old_res, const char *new_res) {
    for (int i = 0; i < n; i++) {
        if (strcmp(q[i].o1, old_res) == 0) {
            strcpy(q[i].o1, new_res);  // Replace operand1
        }
        if (strcmp(q[i].o2, old_res) == 0) {
            strcpy(q[i].o2, new_res);  // Replace operand2
        }
    }
}

int main() {
    FILE *inputFile, *outputFile;

    // Open input file for reading
    inputFile = fopen("quadraple.txt", "r");
    if (inputFile == NULL) {
        printf("Error opening input file!\n");
        return 1;
    }

    // Open output file for writing
    outputFile = fopen("output.txt", "w");
    if (outputFile == NULL) {
        printf("Error opening output file!\n");
        fclose(inputFile);
        return 1;
    }

    Quadruple q[MAX_QUADS];
    int n = 0;

    // Read the quadruples from the input file
    char header[100];
    fgets(header, sizeof(header), inputFile);  // Read and skip the first line (header)

    // Reading the quadruple tuples from the file
    while (fscanf(inputFile, "%s %s %s %s", q[n].op, q[n].o1, q[n].o2, q[n].res) == 4) {
        n++;
    }

    fclose(inputFile);

    // Perform common subexpression elimination
    Quadruple result[MAX_QUADS];
    int result_count = 0;

    for (int i = 0; i < n; i++) {
        char common_res[5];
        if (is_common_subexpression(q, i, common_res)) {
            // If a common subexpression is found, replace its result in the quadruples
            replace_redundant_result(q, n, q[i].res, common_res);
        } else {
            // Add the current quadruple to the result array
            result[result_count++] = q[i];
        }
    }

    // Write the optimized quadruples to output file (single header)
    fprintf(outputFile, "OP\tO1\tO2\tRES\n");  // Print the header once
    for (int i = 0; i < result_count; i++) {
        fprintf(outputFile, "%s\t%s\t%s\t%s\n", result[i].op, result[i].o1, result[i].o2, result[i].res);
    }

    fclose(outputFile);

    printf("Common subexpression elimination completed. Optimized output written to 'output.txt'.\
n");

    return 0;
}
```

QUADRUPLE.TXT

```
Quadruple
OP      O1      O2      RES
*       c       d       t1
*       b       t1      t2
+       a       t2      t3
+       t3      e       t4
*       a       b       t5
*       c       d       t6
+       t5      t6      t7
```

OUTPUT.TXT

```
OP      O1      O2      RES
*       c       d       t1
*       b       t1      t2
+       a       t2      t3
+       t3      e       t4
*       a       b       t5
+       t5      t1      t7
```

EXPERIMENT NO 8
CODE GENERATION

NAME: K P ASHIL
CLASS: S 7 CSE
ROLL NO: 29
DATE: 18/09/2024

CODE:

```
#include <stdio.h>
#include <string.h>
#include<stdlib.h>
void main()
{
    char code[10][30], str[20], opr[10];
    int i = 0,k=0;
    FILE *fp1,*fp2;
    fp1=fopen("input.txt","r");
    printf("file opened\n");
    fp2=fopen("result.txt","w");
    char op,fir[10],sec[10],res[10];
    while(!feof(fp1)){

            fscanf(fp1,"%c%s%s%s\n",&op,fir,sec,res);
            printf("%c %s %s %s\n",op,fir,sec,res);

            switch (op){
                    case '+':
                            strcpy(opr, "ADD ");
                            i=0;
                            break;
                    case '-':
                            strcpy(opr, "SUB");
```

```
                i=0;
                break;
        case '*':
                strcpy(opr, "MUL ");
                i=1;
                break;
        case '/' :
                strcpy(opr, "DIV");
                i=1;
                break;
}
if(strlen(fir)==2)
{
        if (fir[1]=='1')
                fprintf(fp2,"MOV AX,CH\n");
        else if (fir[1]=='2')
                fprintf(fp2,"MOV BX,CL\n");
        else if (fir[1]=='3')
                fprintf(fp2,"MOV CX,DH\n");
        else if (fir[1]=='4')
                fprintf(fp2,"MOV DX,DL\n");
}
else
fprintf(fp2,"MOV AX,[%s]\n", fir);
if(strlen(sec)==2)
{
        if (sec[1]=='1')
                fprintf(fp2,"MOV BX,CH\n");
        else if (sec[1]=='2')
                fprintf(fp2,"MOV BX,CL\n");
        else if (sec[1]=='3')
                fprintf(fp2,"MOV BX,DH\n");
        else if (sec[1]=='4')
                fprintf(fp2,"MOV BX,DL\n");
}
else
        fprintf(fp2,"MOV BX,[%s]\n",sec);
if (i==0)
fprintf(fp2,"%sAX,BX\n", opr);
else
fprintf(fp2,"%s BX\n", opr);
if(strlen(res)==2)
{
        if (res[1]=='1')
                fprintf(fp2,"MOV CH,AX\n");
        else if (res[1]=='2')
                fprintf(fp2,"MOV CL,AX\n");
        else if (res[1]=='3')
                fprintf(fp2,"MOV DH,AX\n");
        else if (res[1]=='4')
                fprintf(fp2,"MOV DL,AX\n");
}
```

```
                k++;

    }
}
```

INPUT.TXT

```
+ a b t1
+ c d t2
/ t1 t2 t3nc
```

RESULT.TXT

```
MOV AX,[a]
MOV BX,[b]
ADD AX,BX
MOV CH,AX
MOV AX,[c]
MOV BX,[d]
ADD AX,BX
MOV CL,AX
MOV AX,CH
MOV BX,CL
DIV BX
```