PCD LAB :

Lexical Analyser Using C :

INPUT:

Add.txt (create a file)

```
void main ( )
{
int a = 10 ;
int b = 10 ;
int c = a + b ;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
const int keywordsLength = 3;
const int operatorsLength = 2;
const int symbolsLength = 5;
char *keywords[3] = {"void", "int", "main"};
char *operator[2] = {"+", "="};
char *symbols[5] = {"(", ")", "{", "}", ";"};
void clearToken(char token[], int size)
{
    for (int i = 0; i < size; i++)
    {
        token[i] = '\0';
    }
}
void processToken(char token[], int size)
{
    if (size == 0)
    {
        return;
    }
    for (int i = 0; i < keywordsLength; ++i)
    {
        if (strcmp(token, keywords[i]) == 0)
        {
            printf("%s  |(keyword)\n", keywords[i]);
            return;
        }
    }
    for (int i = 0; i < operatorsLength; ++i)
    {
        if (strcmp(token, operator[i]) == 0)
        {
            printf("%s    |(operator)\n", operator[i]);
            return;
        }
    }
    for (int i = 0; i < symbolsLength; ++i)
    {
        if (strcmp(token, symbols[i]) == 0)
        {
            printf("%s    |(symbol)\n", symbols[i]);
            return;
        }
    }
}
void logic(char buffer[], int size)
{
    char temp[100];
    int tempIndex = 0;
    for (int i = 0; i < size; i++)
    {
        if (buffer[i] == ' ' || buffer[i] == '\n')
        {
            temp[tempIndex] = '\0';
            processToken(temp, tempIndex);
            clearToken(temp, 100);
            tempIndex = 0;
            continue;
        }
        temp[tempIndex] = buffer[i];
        tempIndex += 1;
    }
}
int main()
{
    FILE *file;
    file = fopen("add.txt", "r");
    int lSize;
    char ch;
    fseek(file, 0L, SEEK_END);
    lSize = ftell(file);
    rewind(file);
    char *buffer;
    buffer = calloc(1, lSize + 1);
    fread(buffer, lSize, 1, file);
    logic(buffer, lSize);
    fclose(file);
    printf("______");
}
```

Lexical Analyser Using LEX Tool :

lexicalanalysis.l

```
%{
#include<stdio.h>
%}

letter [a-zA-Z]
digit [0-9]
operators [+*/=%&|<>-]
specialcharacters [();{}"]

%%

(#include<stdio.h>|void|main|int|float|char|printf|while|do|for|if|else|double|break|continue|sca
nf|switch|case)+ {
    printf(" Keyword ");
}

{letter}({letter}|{digit})* {
    printf(" Variable ");
}

{digit}+ {
    printf(" Number ");
}

{operators}+ {
    printf(" Operator ");
}

{specialcharacters}+ {
    printf(" Specialcharacter ");
}
```

```
%%

int main(int argc,char *argv[]) {
    yyin = fopen(argv[1],"r");
    yylex();
}
```

OUTPUT :

```
lex lexicalanalysis.l
cc lex.yy.c -ll
./a.out aa.c
./a.out aa.c
```

Shift reducer:

INPUT:

```
E
E+E
E
E*E
E
I
i+i*i
```

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


struct stack {
    char s[20];
    int top;
};


struct stack st;


int isempty() {
    return (st.top == 0);
}


void push(char p) {
    st.s[st.top++] = p;
```

```
}


char pop() {
    if (isempty())
        printf("stack empty");
    else
        return st.s[--st.top];
}


void disp() {
    int i;
    for (i = 0; i < st.top; i++)
        printf("%c", st.s[i]);
}


int reduce(int *j, char rp[10][10], int n) {
    int i, t, k;
    char u[10];
    t = st.top - 1;
    for (i = 0; i <= st.top; i++) {
        u[i] = st.s[t];
        u[i + 1] = '\0';
        for (k = 0; k < n; k++) {
            if (strcmp(rp[k], u) == 0) {
                st.top = st.top - i - 1;
                return k;
            }
        }
        t--;
    }
    return 99;
}


int shift(char ip[], int *j) {
    push(ip[(*j)++]);
    disp();
    return 1;
}


int main() {
    int n, i, j = 0, k, h;
    char lp[10];
    char ip[10];
    char rp[10][10];
    st.top = 0;


    printf("\nEnter the number of productions:");
    scanf("%d", &n);


    for (i = 0; i < n; i++) {
        printf("\nEnter the left side of the production %d:", i + 1);
        scanf(" %c", &lp[i]);
        printf("\nEnter the right side of the production %d:", i + 1);
        scanf("%s", rp[i]);
    }


    printf("\nEnter the input:");
    scanf("%s", ip);



printf("===============================================================\n");
    printf("STACK\tINPUT\tOUTPUT\n");

printf("===============================================================\n");


    strcat(ip, "$");
    push('$');


    printf("$\t%s\n", ip);


    while (!(st.s[st.top - 1] == lp[0] && st.s[st.top - 2] == '$' && (j == (strlen(ip) - 1)) && st.top
== 2)) {
        if ((h = reduce(&j, rp, n)) != 99) {
            push(lp[h]);
            disp();
            printf("\t\t\t");
            for (k = j; k < strlen(ip); k++)
                printf("%c", ip[k]);
            printf("\t\t\tReduce %c->%s\n", lp[h], rp[h]);
        } else if (shift(ip, &j)) {
            printf("\t\t\t");
            for (k = j; k < strlen(ip); k++)
                printf("%c", ip[k]);
            printf("\t\t\tShift %c\n", ip[j - 1]);
        }
    }
    disp();
    printf("\t\t\t");
    for (k = j; k < strlen(ip); k++)
        printf("%c", ip[k]);
    printf("\t\t\tAccept\n");


    return 0;
}
```

Frontend :

INPUT:

d=(a-b)+(a-c)+b*c

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

void main() {
    char a[50], b[50];
    int i, j, k, len, ti = 0, count;

    printf("Enter the code: ");
    scanf("%s", a);

    strcpy(b, a);
    len = strlen(a);

    // Handling multiplication and division
    for (i = 0; i < len; i++) {
        if (b[i] == '*' || b[i] == '/') {
            for (j = i - 1; j >= 0 && strchr("+-*/=", b[j]) == NULL; j--);
            k = j + 1;
            count = 0;
            printf("\nt%d=", ti++);
            for (j = j + 1; count < 2 && b[j] != '\0'; j++) {
                if (strchr("+-*/", b[j + 1]) != NULL) // Checking for next operator
                    count++;
                printf("%c", b[j]);
            }
            b[k++] = 't';
            b[k++] = ti - 1 + '0';
            for (j = j, k = k; k < len; k++, j++)
                b[k] = b[j];
            i = 0;
            len = strlen(b); // Update the length of the expression
        }
    }

    // Handling addition and subtraction
    for (i = 0; i < len; i++) {
        if (b[i] == '+' || b[i] == '-') {
            for (j = i - 1; j >= 0 && strchr("+-=", b[j]) == NULL; j--);
            k = j + 1;
            count = 0;
            printf("\nt%d=", ti++);
            for (j = j + 1; count < 2 && b[j] != '\0'; j++) {
                if (strchr("+-", b[j + 1]) != NULL) // Checking for next operator
                    count++;
                printf("%c", b[j]);
            }
            b[k++] = 't';
            b[k++] = ti - 1 + '0';
            for (j = j, k = k; k < len; k++, j++)
                b[k] = b[j];
            len = strlen(b); // Update the length of the expression
        }
    }

    printf("\n%s\n", b);
}
```

<u>Backend :</u>

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

int main() {
    char a[50], id[50], mov[] = "MOVF", mul[] = "MULF", div[] = "DIVF", add[] = "ADDF", sub[] =
"SUBF";
    int i = 0, j = 0, len = 0, s = 0, e = 0, r = 1;
    FILE *fp;
    fp = fopen("out.txt", "w");
```

```
printf("\nEnter the code:");
fgets(a, sizeof(a), stdin);
len = strlen(a);
for (i = 0; i < len; i++) {
    if (a[i] == '=') {
        for (j = i; j < len; j++)
            if (a[j] == 'i') {
                fprintf(fp, "\n%s ", mov);
                fprintf(fp, "%c%c%c,R%d", a[j], a[j + 1], a[j + 2], r++);
            }
    } else if ((a[i] <= '9') && (a[i] >= '0'))
        if ((a[i + 1] <= '9') && (a[i + 1] >= '0'))
            fprintf(fp, "\n%s #%c%c,R%d", mov, a[i], a[i + 1], r++);
}
for (i = len - 1; i >= 0; i--) {
    if (a[i] == '+') {
        fprintf(fp, "\n%s ", add);
        e = a[i - 1];
        e--;
        s = e;
        if (a[i + 1] == 'i')
            fprintf(fp, "R%c,R%d", e, r - 1);
    } else if (a[i] == '-') {
        fprintf(fp, "\n%s ", sub);
        e = a[i - 1];
        e--;
        s = e;
        if (a[i + 1] == 'i')
            fprintf(fp, "R%c,R%c", (a[i + 3] - 1), s);
        else
            fprintf(fp, "R%c,R%d", e, r - 1);
    } else if (a[i] == '*') {
        fprintf(fp, "\n%s ", mul);
        e = a[i - 1];
        e--;
        s = e;
        if (a[i + 1] == 'i')
            fprintf(fp, "R%c,R%c", (a[i + 3] - 1), s);
        else
            fprintf(fp, "R%c,R%d", e, r - 1);
    } else if (a[i] == '/') {
        fprintf(fp, "\n%s ", div);
        e = a[i - 1];
        e--;
        s = e;
        if (a[i + 1] == 'i')
            fprintf(fp, "R%c,R%c", (a[i + 3] - 1), s);
        else
```

```
            fprintf(fp, "R%c,R%d", e, r - 1);
        }
    }
    fprintf(fp, "\n%s R1,id1", mov);
    fclose(fp);
    return 0;
}
```

INPUT:

id1=id2*id3+id4

Symbol Table :

INPUT:

Input.txt (create a file)

```
int a,b=5;
float c;
char d="a";
```

Or

```
int mega , priya = 5 ;
float c ;
char d = "mega" ;
```

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

struct symtab {
    int lineno;
    char var[25], dt[25], val[10];
} sa[20];

int main() {
    int i = 0, max = 0, line = 0;
    char s[25], typ[25], temp[25], gar[] = "garbage";
    FILE *fn;
    fn = fopen("input.txt", "r");
    if (fn == NULL) {
        printf("Error opening file!\n");
        return 1; // Indicate error
    }

    printf("\n\nSYMBOL TABLE MANAGEMENT\n\n");
    printf("Variable\tDatatype\tLine.no.\t\tValue\n");

    while (fscanf(fn, "%s", s) != EOF) {
        if ((strcmp(s, "int") == 0) || (strcmp(s, "float") == 0) || (strcmp(s, "char") == 0)) {
            strcpy(typ, s);
            line++;
            while (strcmp(s, ";") != 0) {
                i++;
                max = i;
                sa[i].lineno = line;
                strcpy(sa[i].dt, typ);
                fscanf(fn, "%s", sa[i].var); // Read variable name directly
                fscanf(fn, "%s", s); // Read next token

                if (strcmp(s, "=") == 0) {
                    fscanf(fn, "%s", sa[i].val); // Read value directly
                    fscanf(fn, "%s", s); // Read next token
                } else {
                    strcpy(sa[i].val, gar);
                }

                if (strcmp(s, ",") == 0)
                    continue;
                else
                    break;
            }
        }
    }

    for (i = 1; i <= max; i++)
        printf("%s\t\t%s\t\t%d\t\t%s\n", sa[i].var, sa[i].dt, sa[i].lineno, sa[i].val);

    fclose(fn);
    return 0; // Indicate success
}
```

Code Optimisation :

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

struct op {
   char l;
   char r[20];
} op[10], pr[10];

int main() { // Changed void main() to int main()

   int a, i, k, j, n, z = 0, m, q;
   char *p, *l, *tem, temp, t; // Changed char *tem to *tem to fix pointer assignment
   char nu[] = "\0";

   printf("\nEnter the number of values: ");
   scanf("%d", &n);

   for (i = 0; i < n; i++) {
      printf("\nLeft: ");
      scanf(" %c", &op[i].l); // Added a space before %c to consume any leading whitespace
      printf("Right: ");
      scanf("%s", op[i].r);
   }

   printf("\nIntermediate code\n");

   for (i = 0; i < n; i++)
      printf("%c = %s\n", op[i].l, op[i].r);

   for (i = 0; i < n; i++) {
      temp = op[i].l;
      p = NULL;
      for (j = 0; j < n; j++) {
         p = strchr(op[j].r, temp);
         if (p) {
            pr[z].l = op[i].l;
            strcpy(pr[z].r, op[i].r);
            z++;
            break;
         }
      }
   }

   printf("\nAfter dead code elimination\n");

   for (k = 0; k < z; k++)
      printf("%c = %s\n", pr[k].l, pr[k].r);
```

```
    for (m = 0; m < z; m++) {
        tem = pr[m].r;
        for (j = m + 1; j < z; j++) {
            p = strstr(tem, pr[j].r);
            if (p) {
                pr[j].l = pr[m].l;
                for (i = 0; i < z; i++) {
                    if (strchr(tem, pr[i].r[0])) { // Changed l to pr[i].r[0] to avoid undefined behavior
                        a = tem - pr[i].r; // Changed l to tem to fix assignment
                        pr[i].r[a] = pr[m].l;
                    }
                }
            }
        }
    }

    printf("\nEliminate common expression\n");

    for (i = 0; i < z; i++)
        printf("%c = %s\n", pr[i].l, pr[i].r);

    for (i = 0; i < z; i++) {
        for (j = i + 1; j < z; j++) {
            q = strcmp(pr[i].r, pr[j].r);
            if ((pr[i].l == pr[j].l) && !q) {
                pr[i].l = '\0';
                strcpy(pr[i].r, nu);
            }
        }
    }

    printf("\nOptimized code\n");

    for (i = 0; i < z; i++)
        if (pr[i].l != '\0')
            printf("%c = %s\n", pr[i].l, pr[i].r);

    return 0; // Added return 0; to indicate successful completion
}
```

Input :

Enter the number of values: 5

Left: a
Right: 10

Left: b
Right: 20

Left: c
Right: a+b

Left: d
Right: a+b

Left: e
Right: c+d