Principles of Complier Construction

(CDCSC14)

PRACTICAL FILE

Submitted by:
Vyom Kaushik

2020UCD2106

CSDS-1(Semester V)

INDEX

PRACTICAL 1

Aim: Implement a program for symbol table using hashing.

Code:

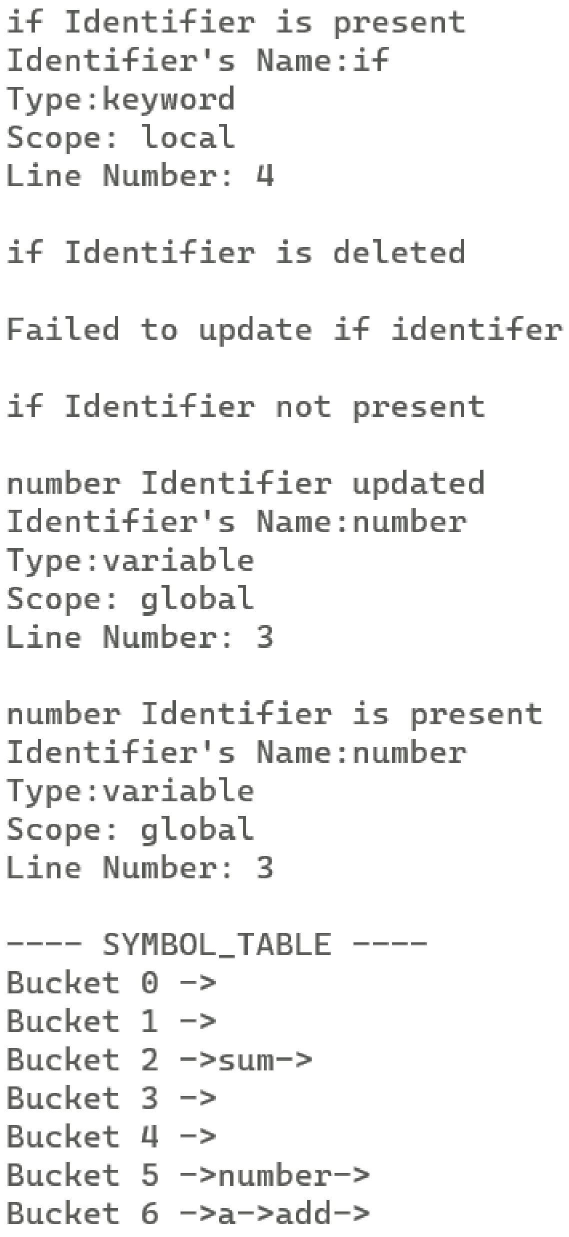
```
#include <bits/stdc++.h>
using namespace std;
class node
{
public:
    string identifier, scope, type;
    int lineNo;
    node *next;
    node(string identifier, string scope, string type, int lineNo){
        this->identifier = identifier;
        this->scope = scope;
        this->type = type;
        this->lineNo = lineNo;
    }
    ~node(){
        if (next != NULL)
        {
            delete next;
        }
    }
    void print(){
        cout << "Identifier's Name:" << identifier
             << endl << "Type:" << type
             << endl <<"Scope: " << scope
             << endl <<"Line Number: " << lineNo << endl;
    }
};
```

```
class symboltable{
    node **table;
    int table_size;
    int hashFn(string key){
        int index = 0;
        int p = 1;
        for (int i = 0; i < key.length(); i++){
            index = index + (key[i] * p) % table_size;
            index = index % table_size;
            p = (p * 27) % table_size;
        }
        return index;
    }

public:
    symboltable(int size = 7){
        table_size = size;
        table = new node *[table_size];
        for (int i = 0; i < table_size; i++){
            table[i] = NULL;
        }
    }
    void insert(string id, string scope, string type, int lineno){
        int index = hashFn(id);
        node *n = new node(id, scope, type, lineno);
        n->next = table[index];
        table[index] = n;
    }
    node *find(string key){
```

```
    int index = hashFn(key);
    node *ptr = table[index];
    while (ptr != NULL){
        if (ptr->identifier == key)
        {
            return ptr;
        }
        ptr = ptr->next;
    }
    return NULL;
}
bool erase(string key){
    int index = hashFn(key);
    node *ptr = table[index];
    if (ptr != NULL){
        if (ptr->identifier == key)
        {
            table[index] = ptr->next;
            return true;
        }
        node *prev = ptr;
        ptr = ptr->next;
        while (ptr != NULL)
        {
            if (ptr->identifier == key)
            {
                prev->next = ptr->next;
                ptr->next = NULL;
                delete ptr;
                return true;
            }
            prev = ptr;
            ptr = ptr->next;
        }
    }
    return false;
}
node *modify(string id, string scope, string type, int lineno){
    int index = hashFn(id);
    node *ptr = table[index];
    while (ptr != NULL){
        if (ptr->identifier == id){
            ptr->scope = scope;
            ptr->type = type;
            ptr->lineNo = lineno;
            return ptr;
        }
        ptr = ptr->next;
    }
    return NULL;
}
void print(){
    for (int i = 0; i < table_size; i++){
        cout << "Bucket " << i << " ->";
        node *ptr = table[i];
        while (ptr != NULL){
            cout << ptr->identifier << "->";
            ptr = ptr->next;
        }
        cout << endl;
        }
    }
};

int main(){
    symboltable s;
    s.insert("if", "local", "keyword", 4);
    s.insert("number", "global", "variable", 2);
    s.insert("add", "global", "function", 1);
    s.insert("sum", "local", "int", 3);
    s.insert("a", "function parameter", "int", 1);
    node *ptr = s.find("if");
    if (ptr != NULL){
        cout << "if Identifier is present\n";
        ptr->print();
    }
    else{
        cout << "if Identifier not present\n";
    }
    if (s.erase("if") == true){
        cout << endl <<"if Identifier is deleted" << endl;
    }
    else{
        cout << endl <<"Failed to delete if identifier" << endl;
    }
    ptr = s.modify("if", "global", "variable", 3);
    if (ptr != NULL){
        cout << endl <<"if Identifier updated" << endl;
        ptr->print();
    }
```

```
else{
   cout << endl <<"Failed to update if identifer" << endl;
}
ptr = s.find("if");
if (ptr != NULL){
   cout << endl <<"if Identifier is present" << endl;
   ptr->print();
}
else{
   cout << endl <<"if Identifier not present" << endl;
}
ptr = s.modify("number", "global", "variable", 3);
if (ptr != NULL){
   cout << endl <<"number Identifier updated" << endl;
   ptr->print();
}
else{
   cout << endl <<"Failed to update number identifer" << endl;
}
ptr = s.find("number");
if (ptr != NULL){
   cout << endl <<"number Identifier is present" << endl;
   ptr->print();
}
else{
   cout << endl <<"number Identifier not present" << endl;
}
cout << endl <<"---- SYMBOL_TABLE ----" << endl;
s.print();
return 0;}
```

Output:



```
if Identifier is present
Identifier's Name:if
Type:keyword
Scope: local
Line Number: 4

if Identifier is deleted

Failed to update if identifer

if Identifier not present

number Identifier updated
Identifier's Name:number
Type:variable
Scope: global
Line Number: 3

number Identifier is present
Identifier's Name:number
Type:variable
Scope: global
Line Number: 3

---- SYMBOL_TABLE ----
Bucket 0 ->
Bucket 1 ->
Bucket 2 ->sum->
Bucket 3 ->
Bucket 4 ->
Bucket 5 ->number->
Bucket 6 ->a->add->
```

Practical 2

Aim: Develop a simple calculator using LEX and YACC tools.

Code:

2.l

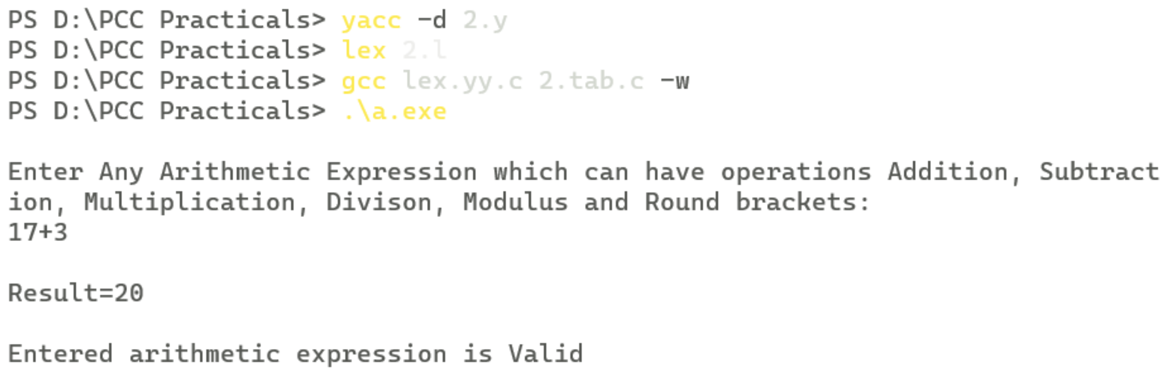
```
%{

#include<stdio.h>
#include "2.tab.h"
extern int yylval;
%}
%%
[0-9]+ {
          yylval=atoi(yytext);
          return NUMBER;
       }
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

2.y

```
%{
    #include<stdio.h>
    int flag=0;
%}
%token NUMBER
%left '+' '-'
%left '*' '/' '%'
%left '(' ')'
%%
ArithmeticExpression: E{
         printf("\nResult=%d\n",$$);
         return 0;
        };
E:E'+'E {$$=$1+$3;}
 |E'-'E {$$=$1-$3;}
 |E'*'E {$$=$1*$3;}
 |E'/'E {$$=$1/$3;}
 |E'%'E {$$=$1%$3;}
 |'('E')' {$$=$2;}
 | NUMBER {$$=$1;}
;
%%
void main()
{
   printf("\nEnter Any Arithmetic Expression which can have operations Addition,
Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");
   yyparse();
  if(flag==0)
   printf("\nEntered arithmetic expression is Valid\n\n");
}
```

```
void yyerror()
{
   printf("\nEntered arithmetic expression is Invalid\n\n");
   flag=1;
}
```

OUTPUT:



```
PS D:\PCC Practicals> yacc -d 2.y
PS D:\PCC Practicals> lex 2.l
PS D:\PCC Practicals> gcc lex.yy.c 2.tab.c -w
PS D:\PCC Practicals> .\a.exe

Enter Any Arithmetic Expression which can have operations Addition, Subtract
ion, Multiplication, Divison, Modulus and Round brackets:
17+3

Result=20

Entered arithmetic expression is Valid
```

Practical 3

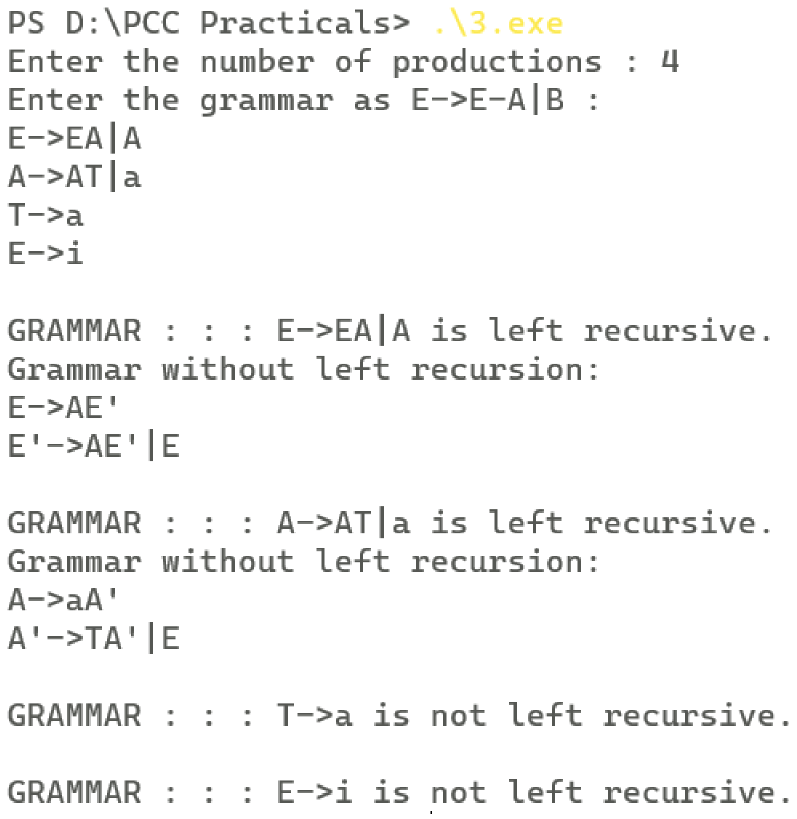
Aim: Write a program to remove left recursion from a context-free grammar

Code:

```
#include<bits/stdc++.h>
#define SIZE 10
int main() {
    char non_terminal;
    char beta, alpha;
    int num;
    char production[10][SIZE];
    int index = 3;
    printf("Enter the number of productions : ");
    scanf("%d", &num);
    printf("Enter the grammar as E->E-A|B : \n");
    for (int i = 0; i < num; i++) {
        scanf("%s", production[i]);
    }
    for (int i = 0; i < num; i++) {
        printf("\nGRAMMAR : : : %s", production[i]);
        non_terminal = production[i][0];
        if (non_terminal == production[i][index]) {
            alpha = production[i][index + 1];
            printf(" is left recursive.\n");
            while (production[i][index] != 0 && production[i][index] !=
'|') index++;
            if (production[i][index] != 0) {
                beta = production[i][index + 1];
                printf("Grammar without left recursion:\n");
                printf("%c->%c%c\'", non_terminal, beta, non_terminal);
                printf("\n%c\'->%c%c\'|E\n", non_terminal, alpha,
non_terminal);
            }
            else printf(" can't be reduced\n");
        }
        else printf(" is not left recursive.\n");
```

```
            index = 3;
        }
    }
```

RESULT:



```
PS D:\PCC Practicals> .\3.exe
Enter the number of productions : 4
Enter the grammar as E->E-A|B :
E->EA|A
A->AT|a
T->a
E->i

GRAMMAR : : : E->EA|A is left recursive.
Grammar without left recursion:
E->AE'
E'->AE'|E

GRAMMAR : : : A->AT|a is left recursive.
Grammar without left recursion:
A->aA'
A'->TA'|E

GRAMMAR : : : T->a is not left recursive.

GRAMMAR : : : E->i is not left recursive.
```

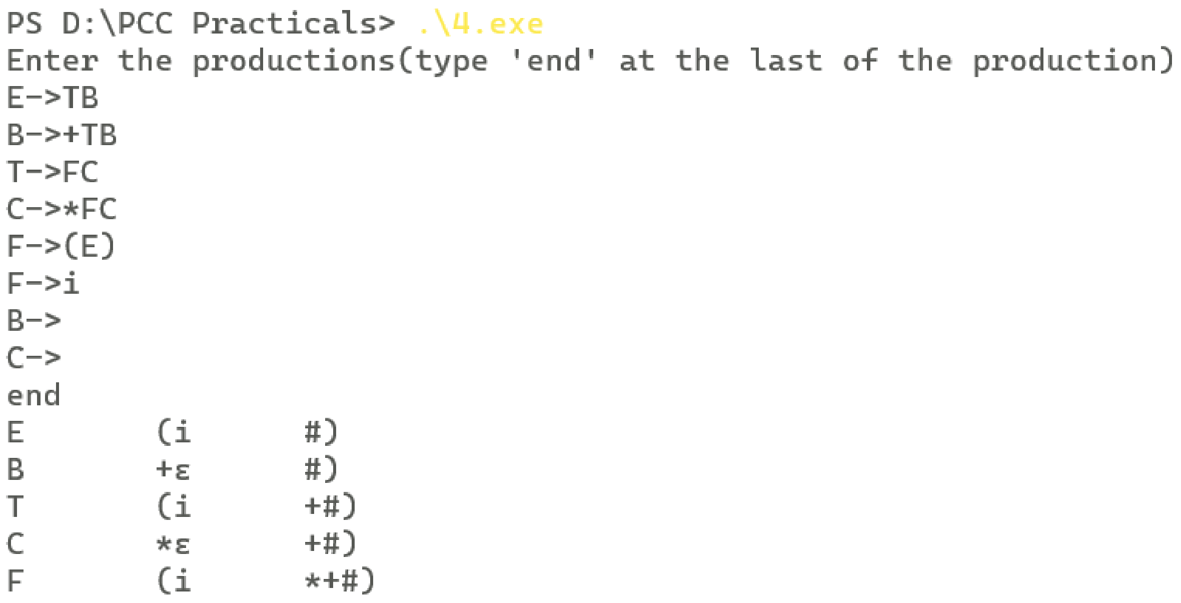
Practical 4

Aim: Write a program to find the first and follow.

Code:

```
#include<iostream>
#include<string.h>
#define max 20
using namespace std;
char prod[max][10];
char ter[10],nt[10];
char first[10][10],follow[10][10];
int eps[10];
int count=0;
int findpos(char ch) {
    int n;
    for(n=0;nt[n]!='\0';n++)
        if(nt[n]==ch) break;
        if(nt[n]=='\0') return 1;
        return n;
}

int IsCap(char c) {
    if(c >= 'A' && c<= 'Z')
        return 1;
    return 0;
}

void add(char *arr,char c) {
    int i,flag=0;
    for(i=0;arr[i]!='\0';i++) {
        if(arr[i] == c) {
            flag=1;
            break;
        }
    }
    if(flag!=1) arr[strlen(arr)] = c;
}

void addarr(char *s1,char *s2) {
    int i,j,flag=99;
    for(i=0;s2[i]!='\0';i++) {
        flag=0;
        for(j=0;;j++) {
            if(s2[i]==s1[j]) {
                flag=1;
                break;
            }
            if(j==strlen(s1) && flag!=1) {
                s1[strlen(s1)] = s2[i];
                break;
            }
        }
    }
}

void addprod(char *s) {
    int i;
    prod[count][0] = s[0];
    for(i=3;s[i]!='\0';i++) {
        if(!IsCap(s[i])) add(ter,s[i]);
        prod[count][i-2] = s[i];
    }
    prod[count][i-2] = '\0';
    add(nt,s[0]);
    count++;
}

void findfirst() {
    int i,j,n,k,e,n1;
    for(i=0;i<count;i++) {
        for(j=0;j<count;j++) {
            n = findpos(prod[j][0]);
            if(prod[j][1] == (char)238) eps[n] = 1;
            else {
                for(k=1,e=1;prod[j][k]!='\0' && e==1;k++) {
                    if(!IsCap(prod[j][k])) {
                        e=0;
                        add(first[n],prod[j][k]);
                    }
                    else {
                        n1 = findpos(prod[j][k]);
                        addarr(first[n],first[n1]);
                        if(eps[n1]==0)
                            e=0;
                    }
                }
                if(e==1) eps[n]=1;
            }
        }
    }
}

void findfollow() {
    int i,j,k,n,e,n1;
    n = findpos(prod[0][0]);
    add(follow[n],'#');
    for(i=0;i<count;i++) {
        for(j=0;j<count;j++) {
            k = strlen(prod[j])-1;
            for(;k>0;k--) {
                if(IsCap(prod[j][k])) {
                    n=findpos(prod[j][k]);
                    if(prod[j][k+1] == '\0')
                    {
                        n1 = findpos(prod[j][0]);
                        addarr(follow[n],follow[n1]);
                    }
                    if(IsCap(prod[j][k+1]))
                    {
                        n1 = findpos(prod[j][k+1]);
                        addarr(follow[n],first[n1]);
                        if(eps[n1]==1)
                        {
                            n1=findpos(prod[j][0]);
                            addarr(follow[n],follow[n1]);
                        }
                    }
                    else if(prod[j][k+1] != '\0')
                        add(follow[n],prod[j][k+1]);
                }
            }
        }
    }
}

int main() {
    char s[max],i;
    cout<<"Enter the productions(type 'end' at the last of the production)\n";
    cin>>s;
    while(strcmp("end",s)) {
        addprod(s);
        cin>>s;
    }
    findfirst();
    findfollow();
    for(i=0;i<strlen(nt);i++) {
        cout<<nt[i]<<"\t";
        cout<<first[i];

        if(eps[i]==1) cout<<((char)238)<<"\t";

        else cout<<"\t";

        cout<<follow[i]<<"\n";

    }

    return 0;;

}
```

RESULT



```
PS D:\PCC Practicals> .\4.exe
Enter the productions(type 'end' at the last of the production)
E->TB
B->+TB
T->FC
C->*FC
F->(E)
F->i
B->
C->
end
E       (i      #)
B       +ε      #)
T       (i      +#)
C       *ε      +#)
F       (i      *+#)
```

Practical 5

Aim: Write a program to implement predictive parsing

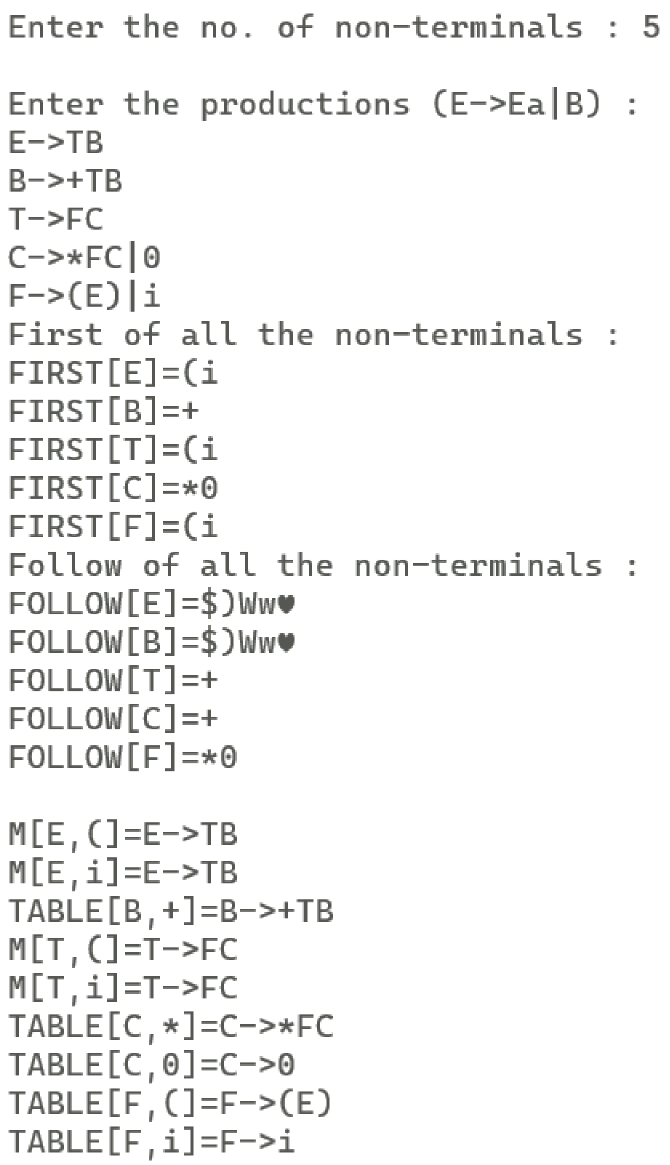
Code

```
#include <stdio.h>

#include <conio.h>

#include <string.h>

int main() {

    char fin[10][20], st[10][20], ft[20][20], fol[20][20];

    int a = 0, e, i, t, b, c, n, k, l = 0, j, s, m, p;

    printf("Enter the no. of non-terminals : ");

    scanf("%d", &n);

    printf("\nEnter the productions (E->Ea|B) : \n");

    for (i = 0; i < n; i++) scanf("%s", st[i]);

    for (i = 0; i < n; i++) fol[i][0] = '\0';

    for (s = 0; s < n; s++) {

        for (i = 0; i < n; i++) {

            j = 3;

            l = 0;

            a = 0;

        l1:

            if (!((st[i][j] > 64) && (st[i][j] < 91))) {

                for (m = 0; m < l; m++) {

                    if (ft[i][m] == st[i][j]) goto s1;

                }

                ft[i][l] = st[i][j];

                l = l + 1;

            s1:

                j = j + 1;

            }

            else {

                if (s > 0) {

                    while (st[i][j] != st[a][0]) {

                        a++;

                    }
                    b = 0;
                    while (ft[a][b] != '\0') {
                        for (m = 0; m < l; m++) {
                            if (ft[i][m] == ft[a][b]) goto s2;
                        }
                        ft[i][l] = ft[a][b];
                        l = l + 1;
                    s2:
                        b = b + 1;
                    }
                }
            }
            while (st[i][j] != '\0') {
                if (st[i][j] == '|') {
                    j = j + 1;
                    goto l1;
                }
                j = j + 1;
            }
            ft[i][l] = '\0';
        }
    }
    printf("First of all the non-terminals : \n");
    for (i = 0; i < n; i++) printf("FIRST[%c]=%s\n", st[i][0], ft[i]);
    fol[0][0] = '$';
    for (i = 0; i < n; i++) {
        k = 0;
        j = 3;
        if (i == 0) l = 1;
        else l = 0;
    k1:
        while ((st[i][0] != st[k][j]) && (k < n)) {
            if (st[k][j] == '\0') {
                k++;
                j = 2;
            }
            j++;
        }
        j = j + 1;
        if (st[i][0] == st[k][j - 1]) {
            if ((st[k][j] != '|') && (st[k][j] != '\0')) {
                a = 0;
                if (!((st[k][j] > 64) && (st[k][j] < 91))) {
                    for (m = 0; m < l; m++) {
                        if (fol[i][m] == st[k][j]) goto q3;
                    }
                    fol[i][l] = st[k][j];
                    l++;
                q3:
                continue;
                }
                else {
                    while (st[k][j] != st[a][0]) {
                        a++;
                    }
                    p = 0;
                    while (ft[a][p] != '\0') {
                        if (ft[a][p] != '@') {
                            for (m = 0; m < l; m++) {
                                if (fol[i][m] == ft[a][p]) goto q2;
                            }
                            fol[i][l] = ft[a][p];
                            l = l + 1;
                        }
                        else e = 1;
                    q2:
                        p++;
                    }
```

```
                if (e == 1) {
                    e = 0;
                    goto a1;
                }
            }
        }
        else {
        a1:
            c = 0;
            a = 0;
            while (st[k][0] != st[a][0]) {
                a++;
            }
            while ((fol[a][c] != '\0') && (st[a][0] != st[i][0])) {
                for (m = 0; m < l; m++) {
                    if (fol[i][m] == fol[a][c])
                        goto q1;
                }
                fol[i][l] = fol[a][c];
                l++;
            q1:
                c++;
            }
        }
        goto k1;
    }
    fol[i][l] = '\0';
}
printf("Follow of all the non-terminals : \n");
for (i = 0; i < n; i++) printf("FOLLOW[%c]=%s\n", st[i][0], fol[i]);
printf("\n");
s = 0;
for (i = 0; i < n; i++) {
    j = 3;
```

```
        while (st[i][j] != '\0') {

            if ((st[i][j - 1] == '|') || (j == 3)) {

                for (p = 0; p <= 2; p++) {

                    fin[s][p] = st[i][p];

                }

                t = j;

                for (p = 3; ((st[i][j] != '|') && (st[i][j] != '\0')); p++)
{

                    fin[s][p] = st[i][j];

                    j++;

                }

                fin[s][p] = '\0';

                if (st[i][k] == '@') {

                    b = 0;

                    a = 0;

                    while (st[a][0] != st[i][0]) {

                        a++;

                    }

                    while (fol[a][b] != '\0') {

                        printf("M[%c,%c]=%s\n", st[i][0], fol[a][b],
fin[s]);

                        b++;

                    }

                }

                else if (!((st[i][t] > 64) && (st[i][t] < 91)))
printf("TABLE[%c,%c]=%s\n", st[i][0], st[i][t], fin[s]);

                else {

                    b = 0;

                    a = 0;

                    while (st[a][0] != st[i][3]) a++;

                    while (ft[a][b] != '\0') {

                        printf("M[%c,%c]=%s\n", st[i][0], ft[a][b],
fin[s]);

                        b++;

                    }

                }
                s++;
            }
            if (st[i][j] == '|') j++;
        }
    }
    getch();
}
```

RESULT:



```
Enter the no. of non-terminals : 5

Enter the productions (E->Ea|B) :
E->TB
B->+TB
T->FC
C->*FC|0
F->(E)|i
First of all the non-terminals :
FIRST[E]=(i
FIRST[B]=+
FIRST[T]=(i
FIRST[C]=*0
FIRST[F]=(i
Follow of all the non-terminals :
FOLLOW[E]=$)Ww♥
FOLLOW[B]=$)Ww♥
FOLLOW[T]=+
FOLLOW[C]=+
FOLLOW[F]=*0

M[E,(]=E->TB
M[E,i]=E->TB
TABLE[B,+]=B->+TB
M[T,(]=T->FC
M[T,i]=T->FC
TABLE[C,*]=C->*FC
TABLE[C,0]=C->0
TABLE[F,(]=F->(E)
TABLE[F,i]=F->i
```

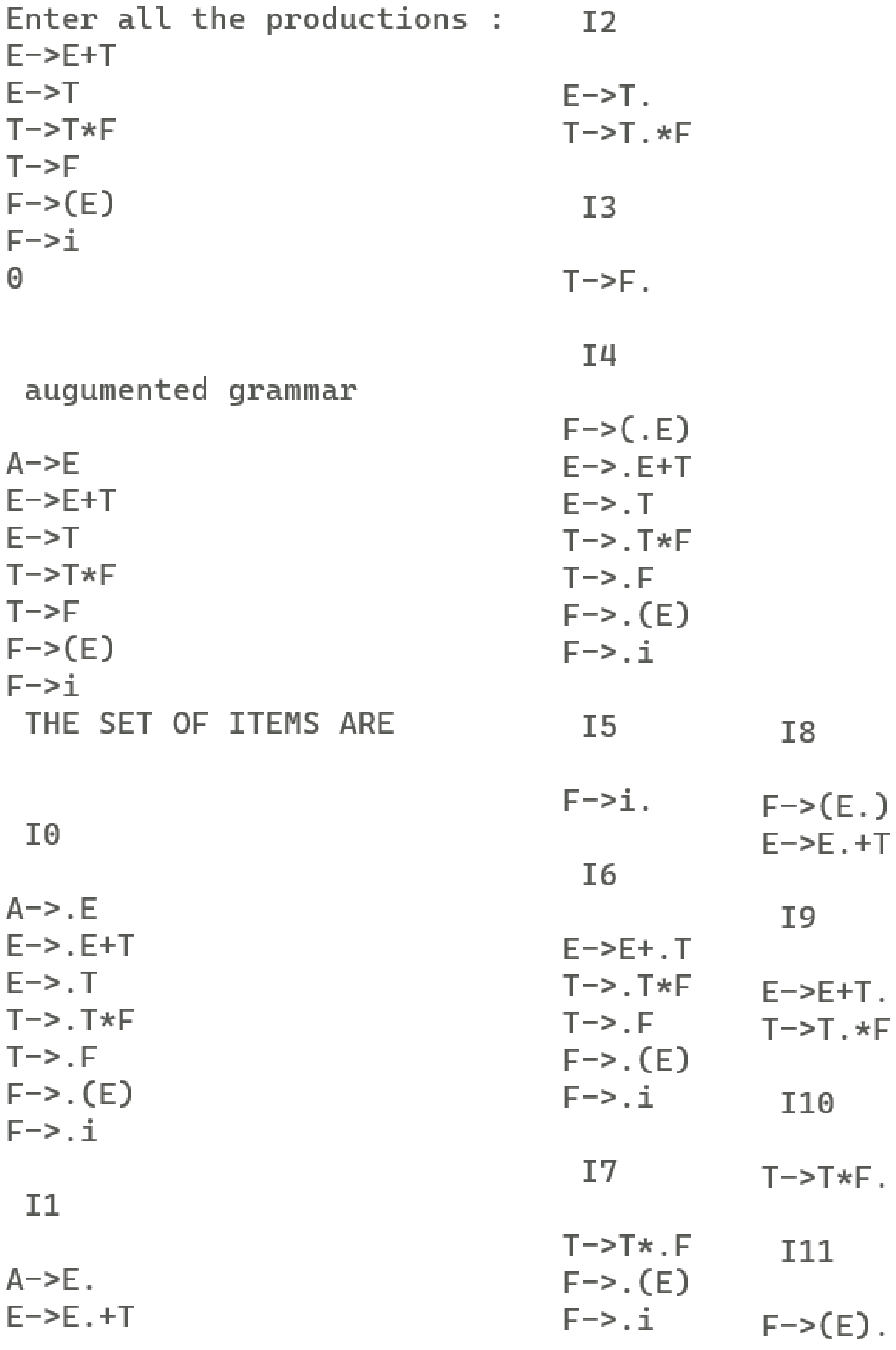
Practical 6

Aim: Write a program to check whether the given grammar is LR (0) or not.

Code

```
#include <iostream>

#include <conio.h>

#include <string.h>

using namespace std;

char prod[20][20], listofvar[26] = "ABCDEFGHIJKLMNOPQR";

int novar = 1, i = 0, j = 0, k = 0, n = 0, m = 0, arr[30];

int noitem = 0;


struct Grammar {

    char lhs;

    char rhs[8];

} g[20], item[20], clos[20][10];


int isvariable(char variable) {

    for (int i = 0; i < novar; i++)

        if (g[i].lhs == variable)

            return i + 1;

    return 0;

}


void findclosure(int z, char a) {

    int n = 0, i = 0, j = 0, k = 0, l = 0;

    for (i = 0; i < arr[z]; i++) {

        for (j = 0; j < strlen(clos[z][i].rhs); j++) {
            if (clos[z][i].rhs[j] == '.' && clos[z][i].rhs[j + 1] == a) {
                clos[noitem][n].lhs = clos[z][i].lhs;
                strcpy(clos[noitem][n].rhs, clos[z][i].rhs);
                char temp = clos[noitem][n].rhs[j];
                clos[noitem][n].rhs[j] = clos[noitem][n].rhs[j + 1];
                clos[noitem][n].rhs[j + 1] = temp;
                n = n + 1;
            }
        }
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < strlen(clos[noitem][i].rhs); j++) {
            if (clos[noitem][i].rhs[j] == '.' &&
isvariable(clos[noitem][i].rhs[j + 1]) > 0) {
                for (k = 0; k < novar; k++) {
                    if (clos[noitem][i].rhs[j + 1] == clos[0][k].lhs) {
                        for (l = 0; l < n; l++)
                        if (clos[noitem][l].lhs == clos[0][k].lhs &&
                            strcmp(clos[noitem][l].rhs, clos[0][k].rhs)==0)
                                break;
                        if (l == n) {
                            clos[noitem][n].lhs = clos[0][k].lhs;
                            strcpy(clos[noitem][n].rhs, clos[0][k].rhs);
                            n = n + 1;
                        }
                    }
                }
            }
        }
    }
    arr[noitem] = n;
    int flag = 0;
    for (i = 0; i < noitem; i++) {
        if (arr[i] == n) {
            for (j = 0; j < arr[i]; j++) {
                int c = 0;
                for (k = 0; k < arr[i]; k++)
                    if (clos[noitem][k].lhs == clos[i][k].lhs &&
                        strcmp(clos[noitem][k].rhs, clos[i][k].rhs) == 0)
                        c = c + 1;
                if (c == arr[i]) {
                    flag = 1;
                    goto exit;
                }
            }
        }
    }
exit:;
    if (flag == 0) arr[noitem++] = n;
}

int main() {
    cout << "Enter all the productions : \n";
    do{
        cin >> prod[i++];
    } while (strcmp(prod[i - 1], "0") != 0);
    for (n = 0; n < i - 1; n++) {
        m = 0;
        j = novar;
        g[novar++].lhs = prod[n][0];
        for (k = 3; k < strlen(prod[n]); k++) {
            if (prod[n][k] != '|') g[j].rhs[m++] = prod[n][k];
            if (prod[n][k] == '|') {
                g[j].rhs[m] = '\0';
                m = 0;
                j = novar;
                g[novar++].lhs = prod[n][0];
            }
        }
    }
    for (i = 0; i < 26; i++) if (!isvariable(listofvar[i])) break;
    g[0].lhs = listofvar[i];
    char temp[2] = {g[1].lhs, '\0'};
    strcat(g[0].rhs, temp);
    cout << "\n\n augumented grammar \n";
    for (i = 0; i < novar; i++) cout << endl << g[i].lhs << "->" <<
g[i].rhs << " ";
    for (i = 0; i < novar; i++) {
        clos[noitem][i].lhs = g[i].lhs;
        strcpy(clos[noitem][i].rhs, g[i].rhs);
        if (strcmp(clos[noitem][i].rhs, "ε") == 0)
strcpy(clos[noitem][i].rhs, ".");
        else {
            for (int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--)
                clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j - 1];
            clos[noitem][i].rhs[0] = '.';

        }

    }

    arr[noitem++] = novar;

    for (int z = 0; z < noitem; z++) {

        char list[10];

        int l = 0;

        for (j = 0; j < arr[z]; j++) {

            for (k = 0; k < strlen(clos[z][j].rhs) - 1; k++) {

                if (clos[z][j].rhs[k] == '.') {

                    for (m = 0; m < l; m++) if (list[m] == clos[z][j].rhs[k
+ 1]) break;

                    if (m == l) list[l++] = clos[z][j].rhs[k + 1];

                }

            }

        }

        for (int x = 0; x < l; x++) findclosure(z, list[x]);

    }

    cout << "\n THE SET OF ITEMS ARE \n\n";

    for (int z = 0; z < noitem; z++) {

        cout << "\n I" << z << "\n\n";

        for (j=0;j<arr[z];j++) cout<<clos[z][j].lhs<<"-
>"<<clos[z][j].rhs<<"\n";

    }

}
```

RESULT



```
Enter all the productions :
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
0


 augumented grammar

A->E
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
 THE SET OF ITEMS ARE


 I0

A->.E
E->.E+T
E->.T
T->.T*F
T->.F
F->.(E)
F->.i

 I1

A->E.
E->E.+T
```

```
 I2

E->T.
T->T.*F

 I3

T->F.

 I4

F->(.E)
E->.E+T
E->.T
T->.T*F
T->.F
F->.(E)
F->.i

 I5

F->i.

 I6

E->E+.T
T->.T*F
T->.F
F->.(E)
F->.i

 I7

T->T*.F
F->.(E)
F->.i
```

```
 I8

F->(E.)
E->E.+T

 I9

E->E+T.
T->T.*F

 I10

T->T*F.

 I11

F->(E).
```

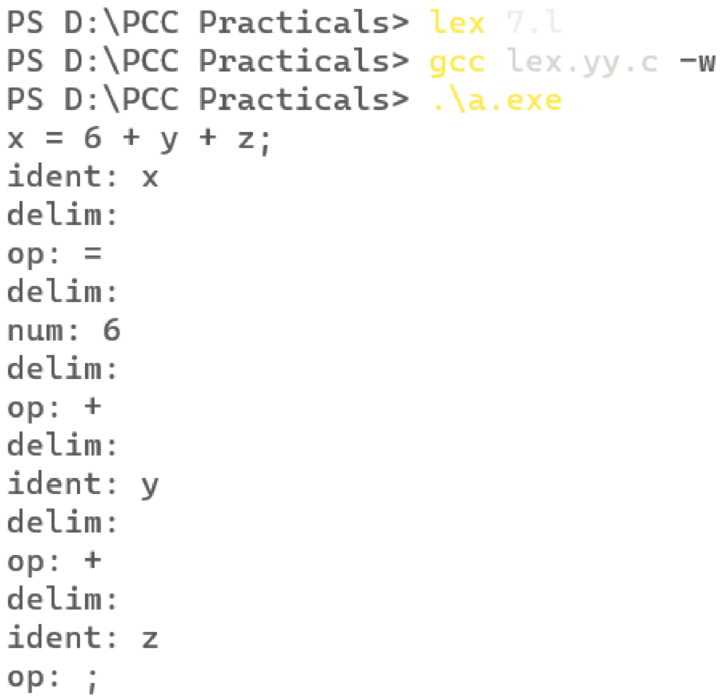
Practical 7

Aim: Write a Lex program to recognize keywords and identifiers in the input "C" program.

Code

```
%option noyywrap

%{

#include<stdio.h>

%}



digit  [0-9]

letter [a-zA-z]

id     {letter}({letter}|{digit})*

delim  [ \t]

operator [+ = - * < > ; <= >= ==]



%%



{digit}+    {printf("num: %s\n" , yytext);}

{id}        {printf("ident: %s\n" , yytext);}

{delim}     {printf("delim: %s\n" , yytext);}

{operator}  {printf("op: %s\n" , yytext);}

.        {printf("other: %s\n", yytext);}



%%



void main()

{

        yylex();
}
```

RESULT



```
PS D:\PCC Practicals> lex 7.l
PS D:\PCC Practicals> gcc lex.yy.c -w
PS D:\PCC Practicals> .\a.exe
x = 6 + y + z;
ident: x
delim:
op: =
delim:
num: 6
delim:
op: +
delim:
ident: y
delim:
op: +
delim:
ident: z
op: ;
```

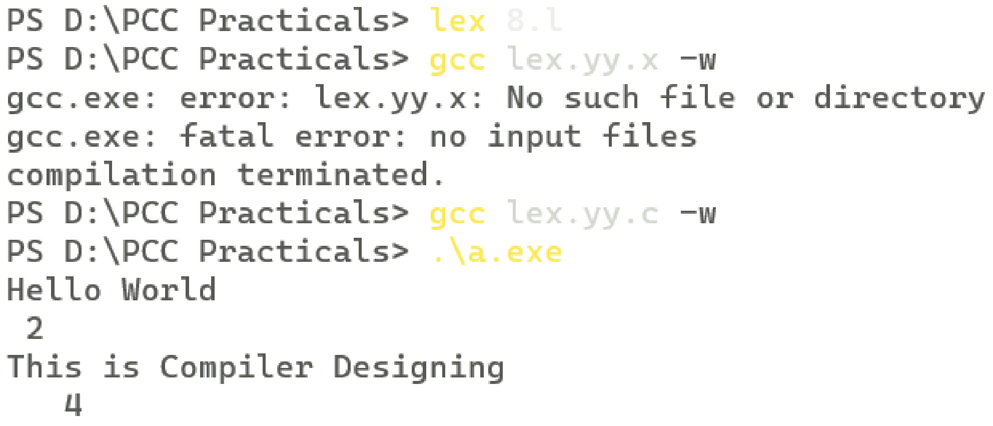
Practical 8

Aim: Write a program to parse an input string as a lexical analyser.

Code

```
%option noyywrap

/*lex program to count number of words*/

%{

#include<stdio.h>

#include<string.h>

int i = 0;

%}


/* Rules Section*/

%%

([a-zA-Z0-9])* {i++;} /* Rule for counting number of words*/


"\n" {printf("%d\n", i); i = 0;}

%%




int main()

{

      // The function that starts the analysis

      yylex();


      return 0;

}
```

RESULT



```
PS D:\PCC Practicals> lex 8.l
PS D:\PCC Practicals> gcc lex.yy.x -w
gcc.exe: error: lex.yy.x: No such file or directory
gcc.exe: fatal error: no input files
compilation terminated.
PS D:\PCC Practicals> gcc lex.yy.c -w
PS D:\PCC Practicals> .\a.exe
Hello World
 2
This is Compiler Designing
   4
```

Practical 9

Aim: Implement a two-pass assembler.

Code

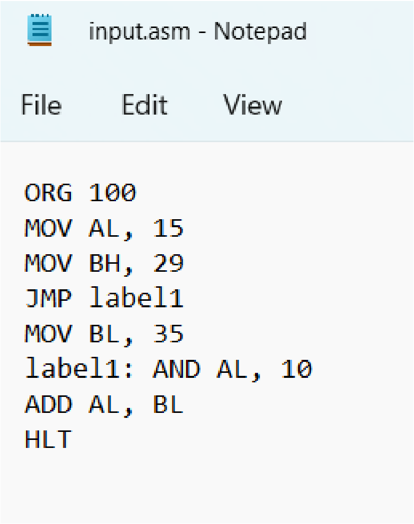
two_pass_assembler.cpp

```
#include <bits/stdc++.h>
using namespace std;

/*
Supported instructions:
ORG
JMP
MOV
ADD
AND
HLT
*/

void mov_hex_value(vector<int> &reg, int start, int len, int val)
{
    for (int i = start; i < start + len; i++)
    {
        reg[i] = val % 16;
        val = val / 16;
    }
}

void add_hex_value(vector<int> &reg, int start, int len, vector<int> &reg2,
int start2, int len2)
{
    if (len != len2)
    {
        cout << "Error" << endl;
        return;
    }

    int carry = 0;

    for (int i = start, j = start2; i < start + len, j < start2 + len2;
i++, j++)
    {
        int val = carry + reg[i] + reg2[j];
        reg[i] = val % 16;
        carry = val / 16;
    }
}

void and_hex_value(vector<int> &reg, int start, int len, int val)
{
    for (int i = start; i < start + len; i++)
    {
        reg[i] = (reg[i] & val) % 16;
        val = val / 16;
    }
}

void show_reg(vector<int> &reg)
{
    for (int i = reg.size() - 1; i >= 0; i--)
    {
        char ch = 'A' + (reg[i] - 10);
        if (reg[i] <= 9)
            cout << reg[i];
        else
            cout << ch;
    }
    cout << endl;
}

int main()
{
    unordered_map<string, int> symbolTable;
    unordered_map<string, string> opCode;

    opCode["JMP"] = "EA", opCode["MOV"] = "B0", opCode["ADD"] = "04";
    opCode["AND"] = "84", opCode["HLT"] = "F4";

    vector<vector<int>> reg(4, vector<int>(4, 0)); // registers

    int starting_address = 0;
    int lines = 0;

    ifstream rdfil;
    rdfil.open("input.asm");

    string line;

    // Pass 1
    while (rdfil >> line)
    {
        if (line == "ORG")
        {
            rdfil >> line;
            starting_address = stoi(line);
        }

        else if (line == "HLT")
        {
            lines++;
            continue;
        }

        else if (line == "JMP")
        {
            rdfil >> line;
            if (symbolTable.find(line) == symbolTable.end())
                symbolTable[line] = -1;
        }

        else if (line == "MOV" or line == "ADD" or line == "AND")
        {
            rdfil >> line;
            rdfil >> line;
        }

        else
        {
            line.pop_back(); // omitting colon
            symbolTable[line] = starting_address + lines;
        }

        lines++;
    }

    cout << "The Symbol Table after Pass 1: " << endl;
    cout << "Label"
         << "\t"
         << "Address" << endl;
    for (auto i = symbolTable.begin(); i != symbolTable.end(); i++)
        cout << i->first << "\t" << i->second << endl;

    cout << endl;
    rdfil.close();

    rdfil.open("input.asm");
    ofstream wtfil("output.txt");

    lines = 0;
    // Pass 2
    while (rdfil >> line)
    {
        wtfil << starting_address + lines << " ";
        if (line == "ORG")
        {
            wtfil << "ORG ";
            rdfil >> line;
            wtfil << line << endl;
        }

        else if (line == "MOV" or line == "ADD" or line == "AND")
        {
            string instruction = line;
            wtfil << opCode[line] << " ";
            rdfil >> line;
            wtfil << line << " ";
            line.pop_back(); // drop comma
            string reg_name = line;
            rdfil >> line;
            wtfil << line << endl;

            int reg_no = reg_name[0] - 'A';
            int len = (reg_name[1] == 'X' ? 4 : 2);
            int start = (reg_name[1] == 'H' ? 2 : 0);

            int literal;
            if (instruction != "ADD")
                literal = stoi(line);

            if (instruction == "MOV")
                mov_hex_value(reg[reg_no], start, len, literal);

            else if (instruction == "AND")
                and_hex_value(reg[reg_no], start, len, literal);

            else
            {
                int reg2_no = line[0] - 'A';
                int len2 = (line[1] == 'X' ? 4 : 2);
                int start2 = (line[1] == 'H' ? 2 : 0);
```

```
                add_hex_value(reg[reg_no], start, len, reg[reg2_no],
start2, len2);
            }
        }

        else if (line == "JMP")
        {
            wtfil << opCode[line] << " ";
            rdfil >> line;
            string label = line;
            wtfil << symbolTable[label] << endl;
            int line_no = symbolTable[label] - starting_address;
            rdfil.close();
            rdfil.open("input.asm");

            int ct = 0;
            while (line_no != ct && getline(rdfil, line))
                ct++;
            rdfil >> line;
        }

        else if (line == "HLT")
        {
            wtfil << opCode[line];
            break;
        }

        else
            wtfil << endl;

        lines++;
    }

    cout << "Output of Pass 2 has been written in output.txt !!!" << endl
         << endl;
    cout << "Here is the value of registers after the program" << endl;
    for (int i = 0; i < 4; i++)
    {
        string str = "";
        str += (char)('A' + i);
        str += "X";
        cout << str << " ";
        show_reg(reg[i]);
    }

    rdfil.close();
    wtfil.close();
    return 0;
}
```

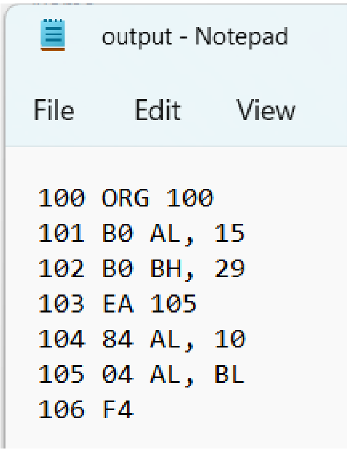
RESULT

input.asm

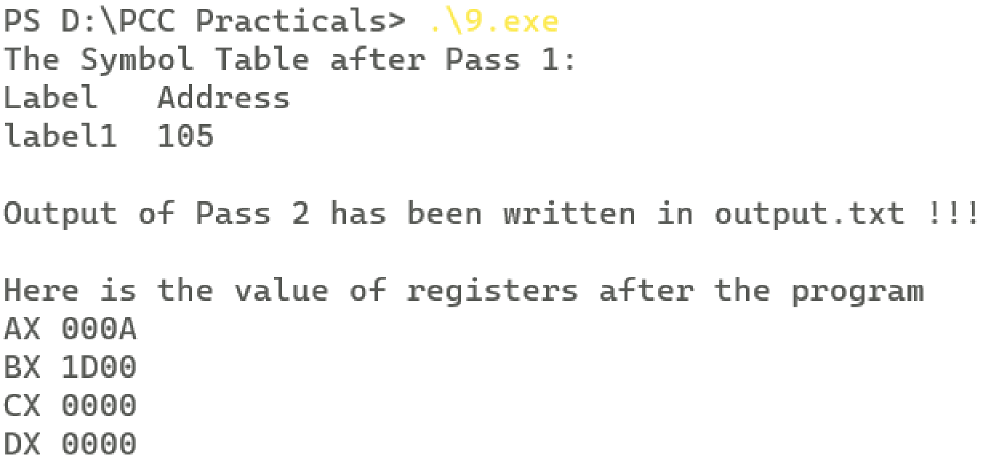


```
ORG 100
MOV AL, 15
MOV BH, 29
JMP label1
MOV BL, 35
label1: AND AL, 10
ADD AL, BL
HLT
```

output.txt



```
100 ORG 100
101 B0 AL, 15
102 B0 BH, 29
103 EA 105
104 84 AL, 10
105 04 AL, BL
106 F4
```



```
PS D:\PCC Practicals> .\9.exe
The Symbol Table after Pass 1:
Label   Address
label1  105

Output of Pass 2 has been written in output.txt !!!

Here is the value of registers after the program
AX 000A
BX 1D00
CX 0000
DX 0000
```

Practical 10

Aim: Write a C program to generate a three-address code for a given expression.

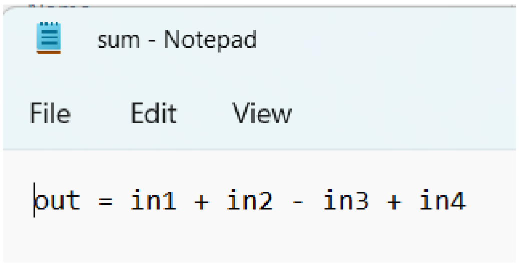
Code

```
#include <iostream>

#include <stdlib.h>

#include <string.h>


using namespace std;


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;

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

    f2 = fopen("out.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);

    return 0;
```

```
}
```

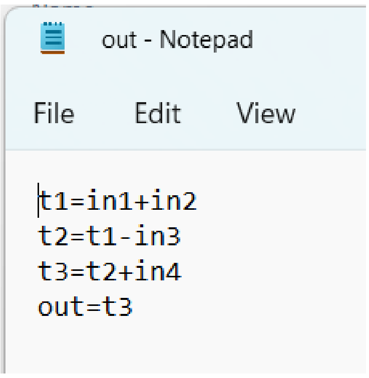
RESULT

sum.txt



```
out = in1 + in2 - in3 + in4
```

out.txt



```
t1=in1+in2
t2=t1-in3
t3=t2+in4
out=t3
```