Compiler Design  
Lab Exercise 11

short line

**Aim: To implement Intermediate Code Generation – Quadruple, Triple, Indirect triple**

**Algorithm:**

The algorithm takes a sequence of three-address statements as input. For each three address statements of the form a:= b op c perform the various actions. These are as follows:

1. Invoke a function getreg to find out the location L where the result of computation b op c should be stored.
2. Consult the address description for y to determine y'. If the value of y currently in memory and register both then prefer the register y' . If the value of y is not already in L then generate the instruction MOV y' , L to place a copy of y in L.
3. Generate the instruction OP z' , L where z' is used to show the current location of z. if z is in both then prefer a register to a memory location. Update the address descriptor of x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all other descriptors.
4. If the current value of y or z have no next uses or not live on exit from the block or in register then alter the register descriptor to indicate that after execution of x : = y op z those register will no longer contain y or z.

**Code:(Language : Python3)**

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

#include<string.h>

void small();

void dove(int i);

int p[5]={0,1,2,3,4},c=1,i,k,l,m,pi;

char sw[5]={'=','-','+','/','\*'},j[20],a[5],b[5],ch[2];

void main()

{

    printf("Enter the expression:");

    scanf("%s",j);

    printf("\tThe Intermediate code is:\n");

    small();

}

void dove(int i)

{

    a[0]=b[0]='\0';

    if(!isdigit(j[i+2])&&!isdigit(j[i-2]))

    {

        a[0]=j[i-1];

        b[0]=j[i+1];

    }

    if(isdigit(j[i+2])){

        a[0]=j[i-1];

        b[0]='t';

        b[1]=j[i+2];

    }

    if(isdigit(j[i-2]))

    {

        b[0]=j[i+1];

        a[0]='t';

        a[1]=j[i-2];

        b[1]='\0';

    }

    if(isdigit(j[i+2]) &&isdigit(j[i-2]))

    {

        a[0]='t';

        b[0]='t';

        a[1]=j[i-2];

        b[1]=j[i+2];

        sprintf(ch,"%d",c);

        j[i+2]=j[i-2]=ch[0];

    }

    if(j[i]=='\*')

        printf("\tt%d=%s\*%s\n",c,a,b);

    if(j[i]=='/')

        printf("\tt%d=%s/%s\n",c,a,b);

    if(j[i]=='+')

        printf("\tt%d=%s+%s\n",c,a,b);

    if(j[i]=='-')

        printf("\tt%d=%s-%s\n",c,a,b);

    if(j[i]=='=')

        printf("\t%c=t%d",j[i-1],--c);

    sprintf(ch,"%d",c);

    j[i]=ch[0];

    c++;

    small();

}

void small()

{

    pi=0;l=0;

    for(i=0;i<strlen(j);i++)

    {

        for(m=0;m<5;m++)

            if(j[i]==sw[m])

                if(pi<=p[m])

                {

                    pi=p[m];

                    l=1;

                    k=i;

                }

    }

    if(l==1)

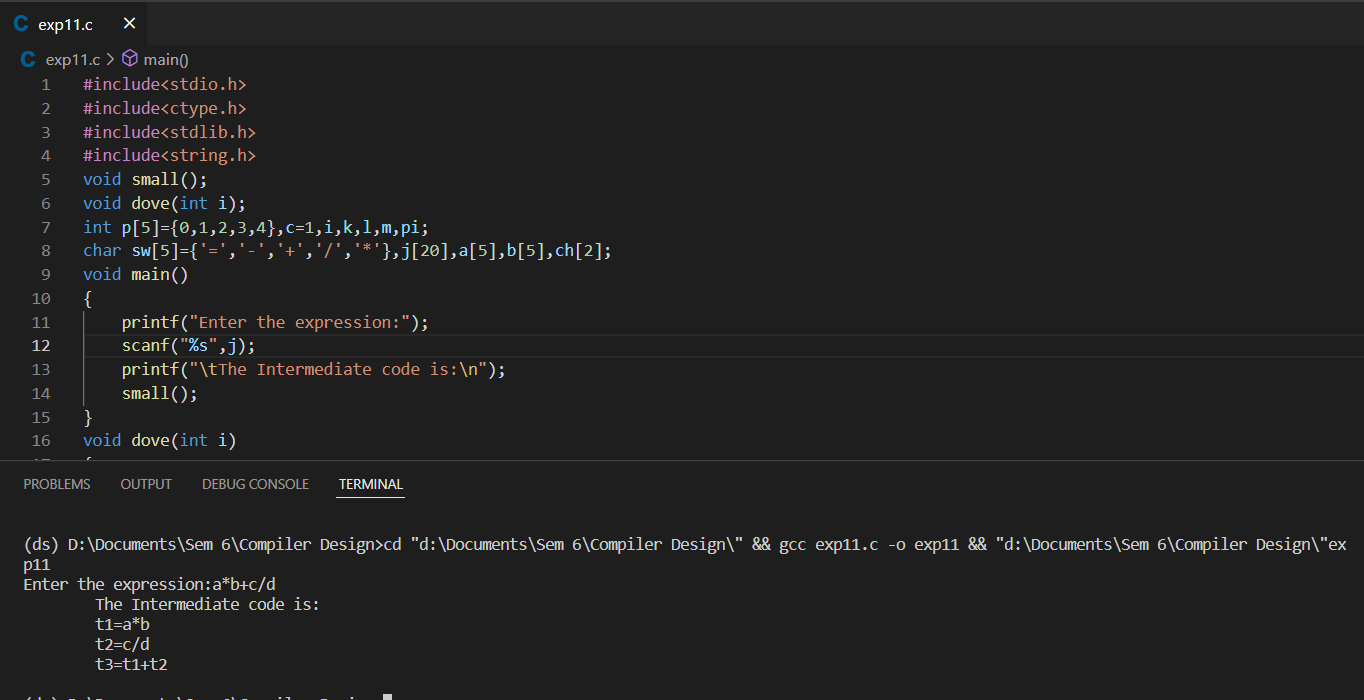
        dove(k);

    else

        exit(0);

}

**Output:**



**Result:** Implementation of Intermediate Code Generation –Quadruple, Triple and Indirect Triple has been completed and verified.