**CD LAB ASSIGNMENT**

*Submitted by:*

*M. Lakshmi Abhigna*

*CSE-F*

*AP21110010343*

**CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

#include<conio.h>

void input();

void output();

void change(int p,char \*res);

void constant();

struct expr

{

char op[2],op1[5],op2[5],res[5];

int flag;

}arr[10];

int n;

void main()

{

input();

constant();

output();

getch();

}

void input()

{

int i;

printf("\n\nEnter the maximum number of  expressions : ");

scanf("%d",&n);

printf("\nEnter the input : \n");

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

{

scanf("%s",arr[i].op);

scanf("%s",arr[i].op1);

scanf("%s",arr[i].op2);

scanf("%s",arr[i].res);

arr[i].flag=0;

}

}

void constant()

{

int i;

int op1,op2,res;

char op,res1[5];

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

{

if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op,"=")==0)

/\*if both digits, store them in variables\*/

{

op1=atoi(arr[i].op1);

op2=atoi(arr[i].op2);

op=arr[i].op[0];

switch(op)

{

case '+':

res=op1+op2;

break;

case '-':

res=op1-op2;

break;

case '\*':

res=op1\*op2;

break;

case '/':

res=op1/op2;

break;

case '=':

res=op1;

break;

}

sprintf(res1,"%d",res);

arr[i].flag=1; /\*eliminate expr and replace any operand below that uses result

of this expr \*/

change(i,res1);

}

}

}

void output()

{

int i=0;

printf("\nOptimized code is : ");

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

{

if(!arr[i].flag)

{

printf("\n%s %s %s %s",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);

}

}

}

void change(int p,char \*res)

{

int i;

for(i=p+1;i<n;i++)

{

if(strcmp(arr[p].res,arr[i].op1)==0)

strcpy(arr[i].op1,res);

else if(strcmp(arr[p].res,arr[i].op2)==0)

strcpy(arr[i].op2,res);

}

}

**CODE EXPLANATION:**

This C code is an optimizer for a simple expression language. It takes a user-defined set of expressions as input, where each expression consists of an operator, two operands, and a result. The program aims to optimize the code by evaluating constant expressions, replacing them with their results, and updating subsequent expressions accordingly.

The `input()` function collects the expressions from the user and stores them in a data structure. The `constant()` function identifies expressions with constant operands, computes their values, and replaces the original expressions with the results. The `change()` function assists in updating subsequent expressions that use the result of the evaluated constant expression. Finally, the `output()` function displays the optimized code, omitting expressions that have been replaced by their constant values.

The program demonstrates a basic constant folding optimization technique, enhancing the efficiency of the provided expressions by precomputing constant values where possible.

**OUTPUT:**

