**CD LAB ASSIGNMENT WEEK-12**

*Submitted by:*

*N. Chaithanya*

*CSE-F*

*AP21110010375*

**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:**

- Data Structure:

- The code defines a structure named ‘expr’ to represent expressions, containing fields for an operator (‘op’), two operands (‘op1’ and ‘op2’), a result (‘res’), and a flag (‘flag’).

- Functions:

- ‘input()’ Function:

- Takes user input for a set of expressions, populating an array of ‘expr’ structures.

- ‘constant()’ Function:

- Identifies expressions with constant operands.

- Evaluates these constant expressions and replaces them with their results.

- Updates subsequent expressions that use the result of the evaluated constant expression.

- ‘change()’ Function:

- A helper function for ‘constant()’.

- Updates subsequent expressions that reference the result of an evaluated constant expression.

- ‘output()’ Function:

- Prints the optimized code, excluding expressions that have been replaced by their constant values.

- Main Function:

- Calls ‘input()’ to gather user input.

- Calls ‘constant()’ to perform constant folding optimization.

- Calls ‘output()’ to display the optimized code.

- Optimization Technique:

- The optimization strategy focuses on identifying expressions with constant values and precomputing these values, thereby eliminating redundant computations and improving code efficiency.

This program serves as a basic optimizer for a custom expression language, demonstrating constant folding as a means to enhance code performance by evaluating expressions with known constant values.

**OUTPUT:**

