LAB ASSIGNMENT - 5

DAG & Code Optimization

Name: Sharan P **Reg.No:** 21BRS1582

PROGRAM

Aim: Generate the three address code and construct the directed acyclic graph using ‘C’.

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

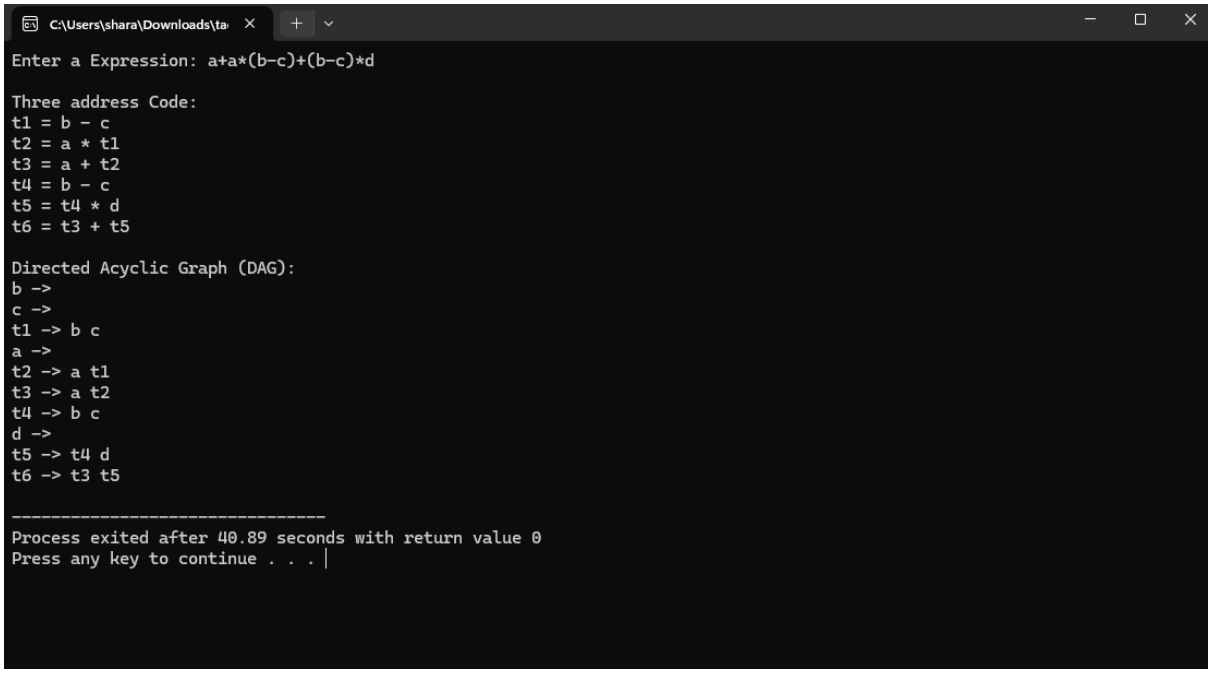
Code:

```
#include <iostream>
#include <vector>
#include <stack>
#include <string>
#include <map>
#include <unordered_map>
using namespace std;
struct Quadruple {
   string op;
   string arg1;
   string arg2;
   string result;
};
struct DAGNode {
   string label;
   vector<int> children;
};
vector<Quadruple> generate3AddressCode(const string& expression) {
   vector<Quadruple> quadruples;
   stack<char> operators;
   stack<string> operands;
   map<char, int> precedence;
   precedence['+'] = precedence['-'] = 1;
   precedence['*'] = precedence['/'] = precedence['%'] = 2;
   precedence['^'] = 3;

   for (char c : expression) {
```

```
        if (c == '(') {
            operators.push(c);
        } else if (isdigit(c) || isalpha(c)) {
            operands.push(string(1, c));
        } else if (c == ')') {
            while (!operators.empty() && operators.top() != '(') {
                char op = operators.top();
                operators.pop();
                string arg2 = operands.top();
                operands.pop();
                string arg1 = operands.top();
                operands.pop();
                string result = "t" + to_string(quadruples.size() + 1);
                quadruples.push_back({string(1, op), arg1, arg2, result});
                operands.push(result);
            }
            operators.pop();
        } else {
            while (!operators.empty() && precedence[operators.top()] >= precedence[c]) {
                char op = operators.top();
                operators.pop();
                string arg2 = operands.top();
                operands.pop();
                string arg1 = operands.top();
                operands.pop();
                string result = "t" + to_string(quadruples.size() + 1);
                quadruples.push_back({string(1, op), arg1, arg2, result});
                operands.push(result);
            }
            operators.push(c);
        }
    }
    while (!operators.empty()) {
        char op = operators.top();
        operators.pop();
        string arg2 = operands.top();
        operands.pop();
        string arg1 = operands.top();
        operands.pop();
        string result = "t" + to_string(quadruples.size() + 1);
        quadruples.push_back({string(1, op), arg1, arg2, result});
        operands.push(result);
    }

    return quadruples;
}
vector<DAGNode> constructDAG(const vector<Quadruple>& quadruples) {
    unordered_map<string, int> nodeMap;
    vector<DAGNode> DAG;

    for (const auto& quad : quadruples) {
```

```
        DAGNode newNode;
        newNode.label = quad.result;
        if (nodeMap.find(quad.arg1) == nodeMap.end()) {
            DAGNode arg1Node;
            arg1Node.label = quad.arg1;
            DAG.push_back(arg1Node);
            nodeMap[quad.arg1] = DAG.size() - 1;
        }
        if (nodeMap.find(quad.arg2) == nodeMap.end()) {
            DAGNode arg2Node;
            arg2Node.label = quad.arg2;
            DAG.push_back(arg2Node);
            nodeMap[quad.arg2] = DAG.size() - 1;
        }
        newNode.children.push_back(nodeMap[quad.arg1]);
        newNode.children.push_back(nodeMap[quad.arg2]);
        DAG.push_back(newNode);
        nodeMap[quad.result] = DAG.size() - 1;
    }

    return DAG;
}
void printDAG(const vector<DAGNode>& DAG) {
    cout << "Directed Acyclic Graph (DAG):\n";
    for (size_t i = 0; i < DAG.size(); ++i) {
        cout << DAG[i].label << " -> ";
        for (int child : DAG[i].children) {
            cout << DAG[child].label << " ";
        }
        cout << endl;
    }
}
int main() {
            cout<<"Enter a Expression: ";
            string expression;cin>>expression;
            cout<<"\nThree address Code: \n";
    vector<Quadruple> quadruples = generate3AddressCode(expression);
    for (const auto& quad : quadruples) {
        cout << quad.result << " = " << quad.arg1 << " " << quad.op << " " << quad.arg2 << endl;
    }
    cout << endl;
    vector<DAGNode> DAG = constructDAG(quadruples);
    printDAG(DAG);
    return 0;
}
```

Output Screenshot:



Result:

We have generated the three address code and constructed the directed acyclic graph using ‘c’.

END OF PROGRAM 1

PROGRAM 2

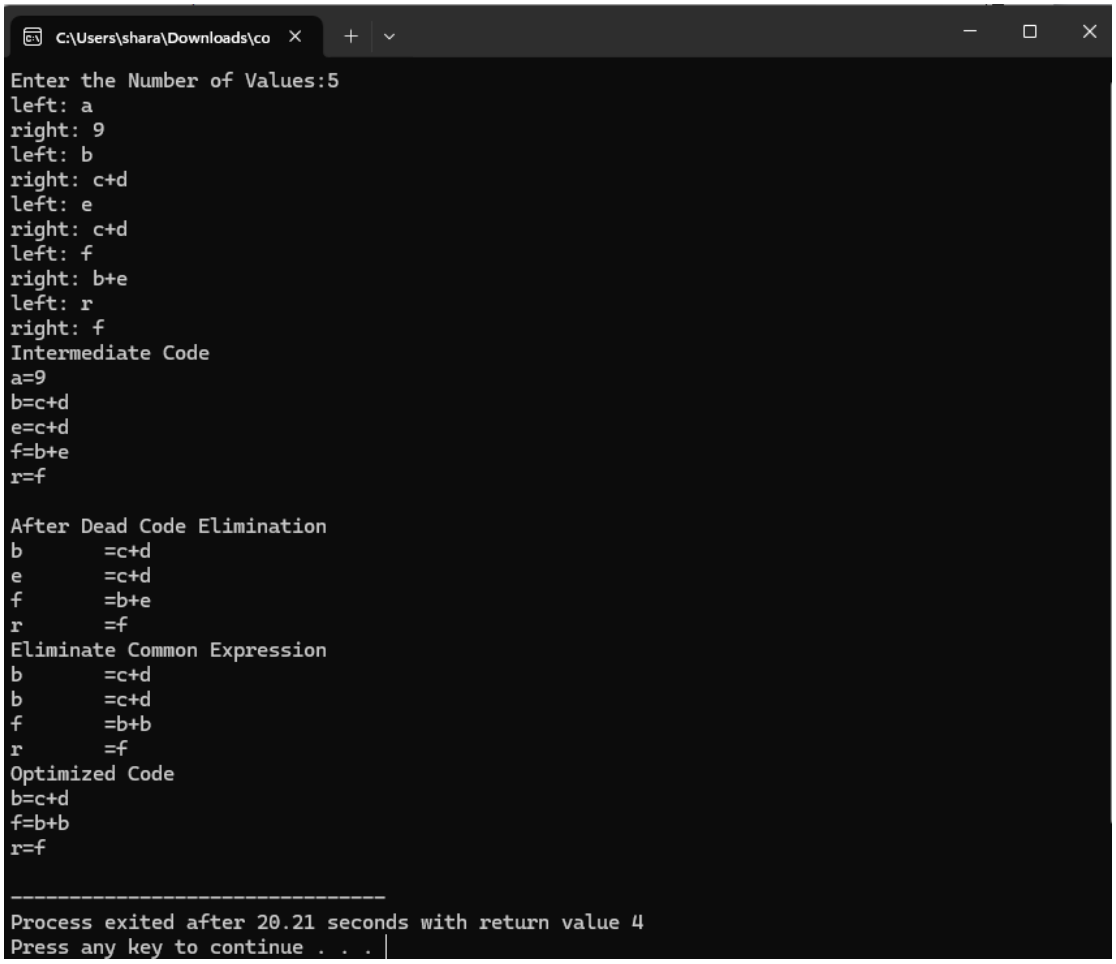
Aim: To implement Simple Code Optimization Techniques.

Code:

```
#include<stdio.h>
#include<string.h>
struct op
{
        char l;
        char r[20];
}
op[10],pr[10];
void main()
{
        int a,i,k,j,n,z=0,m,q;
        char *p,*l;
        char temp,t;
        char *tem;
        printf("Enter the Number of Values:");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("left: ");
                scanf(" %c",&op[i].l);
                printf("right: ");
                scanf(" %s",&op[i].r);
        }
        printf("Intermediate Code\n") ;
        for(i=0;i<n;i++){
                printf("%c=",op[i].l);
                printf("%s\n",op[i].r);
        }
        for(i=0;i<n-1;i++){
                temp=op[i].l;
                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++;
                        }
                }
        }
        pr[z].l=op[n-1].l;
        strcpy(pr[z].r,op[n-1].r);
```

```
	z++;
	printf("\nAfter Dead Code Elimination\n");
	for(k=0;k<z;k++){
		printf("%c\t=",pr[k].l);
		printf("%s\n",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){
				t=pr[j].l;
				pr[j].l=pr[m].l;
				for(i=0;i<z;i++){
					l=strchr(pr[i].r,t) ;
					if(l){
						a=l-pr[i].r;
						//printf("pos: %d\n",a);
						pr[i].r[a]=pr[m].l;
						}
				}
			}
		}
	}
	printf("Eliminate Common Expression\n");
	for(i=0;i<z;i++){
		printf("%c\t=",pr[i].l);
		printf("%s\n",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';
			}
		}
	}
	printf("Optimized Code\n");
	for(i=0;i<z;i++){
		if(pr[i].l!='\0'){
			printf("%c=",pr[i].l);
			printf("%s\n",pr[i].r);
		}
	}
}
```

Output:



Result:

We have implemented simple code optimization techniques like dead-code removal and common expression eliminations.

END OF REPORT
