CD Lab Practical 8

Name : Chetan Pardhi

Roll No. : 22

Batch : B1

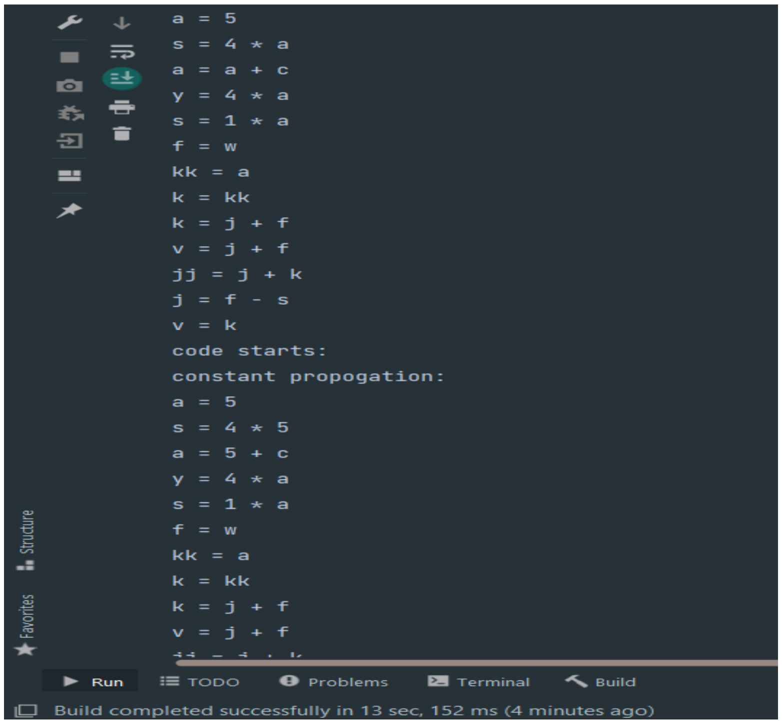
Aim : Write a code to implement Local optimization techniques until no further optimization is possible for the given three address code.

Code :

```
package CD_Lab8;

import java.util.*;
import java.io.*;
import java.io.File;
import java.util.Scanner;

class Expression{

    String left, operator;
    String r1, r2;
    boolean isUnary;

    public Expression(String left, String r1){
        this.left = left;
        this.r1 = r1;
        this.isUnary = true;
    }
    public Expression(String left, String r1, String operator, String r2){
        this.left = left;
        this.r1 = r1;
        this.operator = operator;
        this.r2 = r2;
        this.isUnary = false;
    }
    public String toString(){
        return left + " = " + r1 + ((isUnary)?"" : ( " "+ operator + " " +
r2));
    }
}

public class CD_Lab_Prac8{
    public static void main(String [] args){

        Expression[] exps = new Expression[13];
        // new Expression(")
        exps[0] = new Expression("a","5");
        exps[1] = new Expression("s","4","*","a");
        exps[2] = new Expression("a","a","+","c");
        exps[3] = new Expression("y","4","*","a");
        exps[4] = new Expression("s","1","*","a");
        exps[5] = new Expression("f","w");

        exps[6] = new Expression("kk","a");
        exps[7] = new Expression("k","kk");
        exps[8] = new Expression("k", "j", "+", "f");

        exps[9] = new Expression("v", "j", "+", "f");
        exps[10] = new Expression("jj", "j", "+", "k");
        exps[11] = new Expression("j", "f", "-", "s");
        exps[12] = new Expression("v", "k");


        int expsLen = exps.length;
        printExp(exps);
        System.out.println("code starts:");
        System.out.println("constant propogation:");


        for (int i = 0; i < expsLen; i++){
            Expression currExp = exps[i];

            if(currExp.isUnary && isInt(currExp.r1)){
                for (int inner = i+1; inner < expsLen; inner++){
                    Expression innerExp = exps[inner];
                    if(innerExp.r1.equals(currExp.left)){
                        innerExp.r1 = currExp.r1;
                    }
                    if(!innerExp.isUnary &&
innerExp.r2.equals(currExp.left)){
                        innerExp.r2 = currExp.r1;
                    }
                    if(innerExp.left.equals(currExp.left)){
                        break;
                    }
                }
            }
        }
        printExp(exps);
        // constant folding

        System.out.println("Constant Folding:");
        for (int i = 0; i < expsLen; i++){
            Expression currExp = exps[i];

            if(!currExp.isUnary && isInt(currExp.r1) && isInt(currExp.r2))
{
                int val1 = Integer.parseInt(currExp.r1);
                int val2 = Integer.parseInt(currExp.r2);

                int val = getCalculated(currExp.operator, val1, val2);
                currExp.isUnary = true;
                currExp.r1 = "" + val;
                currExp.r2 = null;
                currExp.operator = null;
            }
        }

        printExp(exps);

        for (int i = 0; i < expsLen; i++){
            Expression currExp = exps[i];

            if(!currExp.isUnary){
                if(isInt(currExp.r1)){
                    int v1 = Integer.parseInt(currExp.r1);
                    if (v1 == 1){
                        if(currExp.operator.equals("*")){
                            currExp.r1  = currExp.r2;
                            currExp.operator = null;
                            currExp.r2 = null;
                            currExp.isUnary = true;
                        }
                    }
                    else if(v1 == 0){
                        if(currExp.operator.equals("+") &&
currExp.operator.equals("-")){
                            currExp.r1  = currExp.r2;
                            currExp.operator = null;
                            currExp.r2 = null;
                            currExp.isUnary = true;

                        }
                        else if(currExp.operator.equals("*")){
                            currExp.r1="0";
                            currExp.operator = null;
                            currExp.r2 = null;
                            currExp.isUnary = true;
                        }
                    }
                }
                else if(isInt(currExp.r2)){
                    int v2 = Integer.parseInt(currExp.r2);
                    if (v2 == 1){
                        if(currExp.operator.equals("*")){
                            currExp.operator = null;
                            currExp.r2 = null;
                            currExp.isUnary = true;
                        }
                    }
                    else if(v2 == 0){
                        if(currExp.operator.equals("+") &&
currExp.operator.equals("-")){
                            currExp.operator = null;
                            currExp.r2 = null;
                            currExp.isUnary = true;
                        }
                        else if(currExp.operator.equals("*")){
                            currExp.r1="0";
                            currExp.operator = null;
                            currExp.r2 = null;
                            currExp.isUnary = true;
                        }
                    }
                }
            }

        }
        System.out.println("Arithmetic removal:");
        printExp(exps);


        for (int i = 0; i < expsLen; i++){
            Expression currExp = exps[i];

            if(currExp.isUnary && !isInt(currExp.r1)){
                for (int inner = i+1; inner < expsLen; inner++){
                    Expression innerExp = exps[inner];
                    if(innerExp.r1.equals(currExp.left)){
                        innerExp.r1 = currExp.r1;
                    }
                    if(!innerExp.isUnary &&
innerExp.r2.equals(currExp.left)){
                        innerExp.r2 = currExp.r1;
                    }
                    if(innerExp.left.equals(currExp.left)){
                        break;
                    }
                }

            }

        }
        System.out.println("copy propogation: ");
        printExp(exps);
```

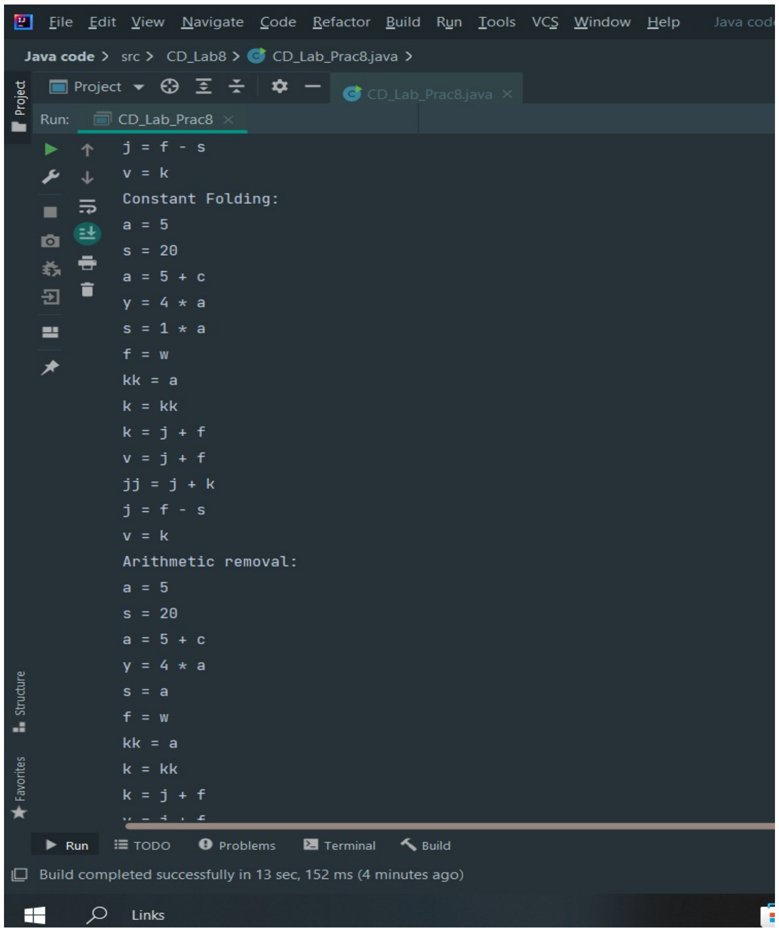
```java
        for(Expression exp: exps) {
            if(exp.r2 == null){
                exp.isUnary = true;
            }
        }


        boolean[] toRemove = new boolean[expsLen];

        for (int i = 0; i < expsLen; i++){
            Expression currExp = exps[i];
            boolean shouldRemove = true;

            for (int inner = i+1; inner < expsLen; inner++){
                Expression innerExp = exps[inner];

                if(innerExp.r1.equals(currExp.left) || (!innerExp.isUnary
&& innerExp.r2.equals(currExp.left))){
                    shouldRemove = false;
                    break;
                }
            }
            toRemove [i] = shouldRemove;


        }


        System.out.println("dead code removal: ");
        printExp(exps);

        for(boolean b : toRemove){
            System.out.println(b);
        }



        System.out.println("Finally");
        printExp(exps, toRemove);




    }
    public static boolean isInt(String str){
        try{
            Integer.parseInt(str);
            return true;
        }
        catch(Exception e) {} return false;
    }

    public static int getCalculated(String operator, int v1, int v2){
        if(operator.equals("+")) return v1+v2;
        if(operator.equals("-")) return v1-v2;
        if(operator.equals("*")) return v1*v2;
        return v1/v2;
    }

    public static void printExp(Expression[] exps){
        for(Expression exp : exps) {
            System.out.println(exp);
        }
    }

    public static void printExp(Expression[] exps, boolean[] toRemove){
        for(int i = 0; i < exps.length; i++){
            if(i == 7) continue;
            if(!toRemove[i]) System.out.println(exps[i]);
        }
    }

    public static boolean contains1(List<String> one, List<String> two ){
        for(String o : one){
            for (String t : two){
                if(o.equals(t)){
                    return true;
                }
            }
        }
        return false;
    }
}
```

Output :



```
a = 5
s = 4 * a
a = a + c
y = 4 * a
s = 1 * a
f = w
kk = a
k = kk
k = j + f
v = j + f
jj = j + k
j = f - s
v = k
code starts:
constant propogation:
a = 5
s = 4 * 5
a = 5 + c
y = 4 * a
s = 1 * a
f = w
kk = a
k = kk
k = j + f
v = j + f
```

Build completed successfully in 13 sec, 152 ms (4 minutes ago)

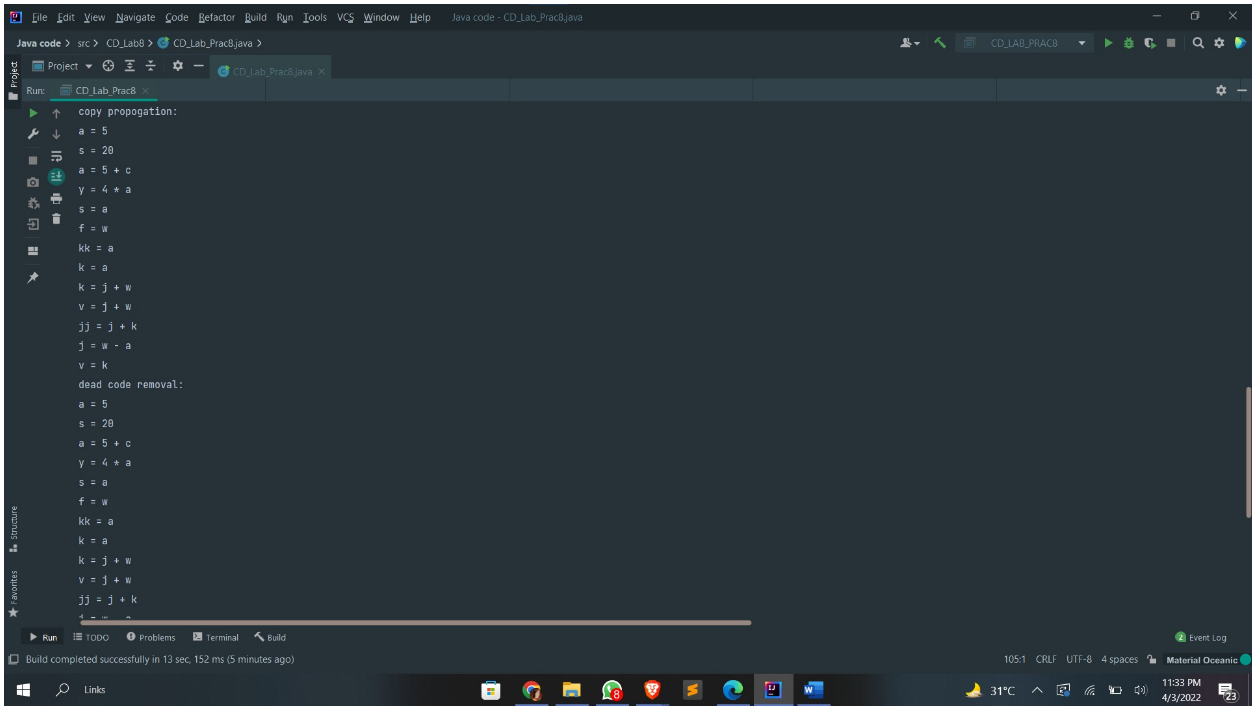


Java code > src > CD_Lab8 > CD_Lab_Prac8.java >

Run: CD_Lab_Prac8

```
j = f - s
v = k
Constant Folding:
a = 5
s = 20
a = 5 + c
y = 4 * a
s = 1 * a
f = w
kk = a
k = kk
k = j + f
v = j + f
jj = j + k
j = f - s
v = k
Arithmetic removal:
a = 5
s = 20
a = 5 + c
y = 4 * a
s = a
f = w
kk = a
k = kk
k = j + f
```

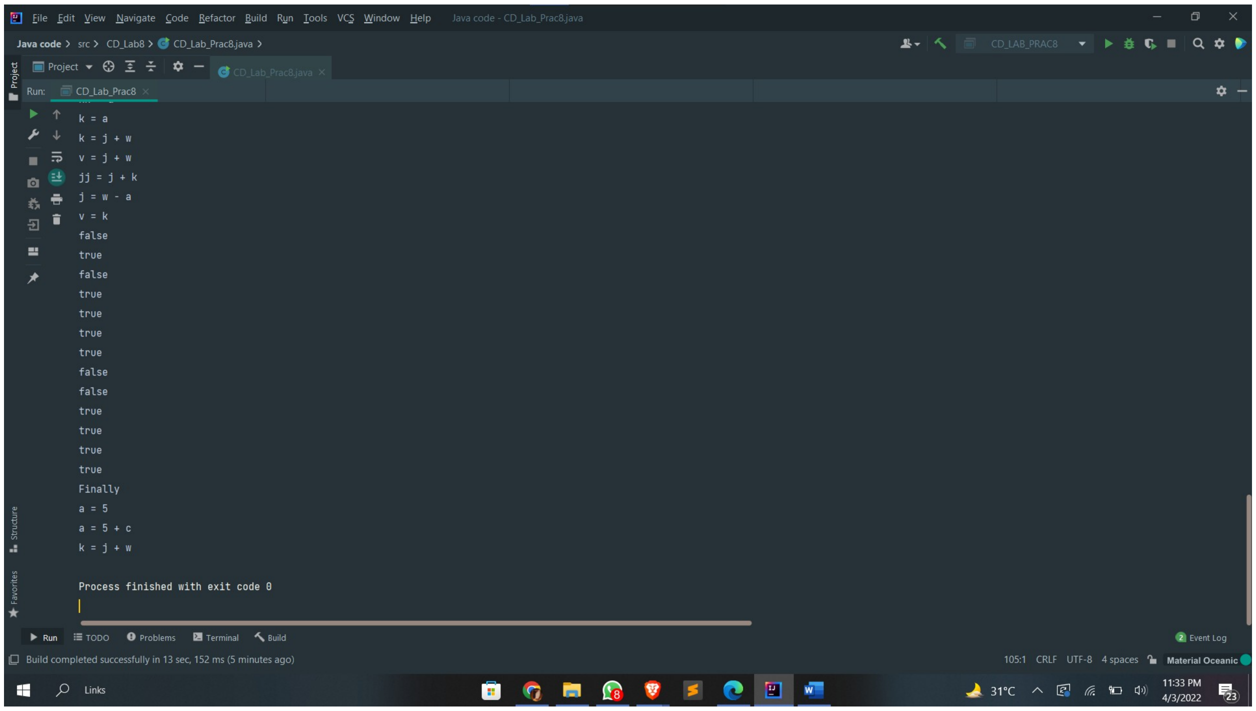
Build completed successfully in 13 sec, 152 ms (4 minutes ago)



```
copy propogation:
a = 5
s = 20
a = 5 + c
y = 4 * a
s = a
f = w
kk = a
k = a
k = j + w
v = j + w
jj = j + k
j = w - a
v = k
dead code removal:
a = 5
s = 20
a = 5 + c
y = 4 * a
s = a
f = w
kk = a
k = a
k = j + w
v = j + w
jj = j + k
```

Build completed successfully in 13 sec, 152 ms (5 minutes ago)

Java code - CD_Lab_Prac8.java
Java code > src > CD_Lab8 > CD_Lab_Prac8.java
Run: CD_Lab_Prac8
11:33 PM
4/3/2022



```
k = a
k = j + w
v = j + w
jj = j + k
j = w - a
v = k
false
true
false
true
true
true
true
false
false
true
true
true
true
Finally
a = 5
a = 5 + c
k = j + w

Process finished with exit code 0
```

Build completed successfully in 13 sec, 152 ms (5 minutes ago)

Java code - CD_Lab_Prac8.java
Java code > src > CD_Lab8 > CD_Lab_Prac8.java
Run: CD_Lab_Prac8
11:33 PM
4/3/2022