**EXP NO : 11 DATE :**

**IMPLEMENT CODE OPTIMIZATION TECHNIQUES LIKE DEAD CODE AND COMMON EXPRESSION ELIMINATION**

**AIM:**

The aim is to implement code optimization techniques such as Dead Code Elimination (DCE) and Common Subexpression Elimination (CSE) on an intermediate representation of a program (such as Three-Address Code (TAC)). These optimization techniques help reduce the size of the code, improve runtime performance, and eliminate redundant computations during the compilation process.

**ALGORITHM:**

* Start
* Create the input file which contains three address code.
* Open the file in read mode.
* If the file pointer returns NULL, exit the program else go to 5.
* Scan the input symbol from left to right.
* Store the first expression in a string.
* Compare the string with the other expressions in the file.
* If there is a match, remove the expression from the input file.
* Perform these steps 5-8 for all the input symbols in the file.
* Scan the input symbol from the file from left to right.
* Get the operand before the operator from the three address code.
* Check whether the operand is used in any other expression in the three address code.
* If the operand is not used, then eliminate the complete expression from the three address code else go to 14.
* Perform steps 11 to 13 for all the operands in the three address code till end of the file is reached.
* Stop.

**PROGRAM:**

#include <stdio.h> #include <stdlib.h> #include <string.h>

#define MAX 100 typedef struct {

char lhs[20], rhs[50];

} TAC;

int isUsed(TAC tac[], int total, char \*var, int current) { for (int i = current + 1; i < total; i++) {

if (strstr(tac[i].rhs, var)) return 1;

}

return 0;

}

void replaceVar(char \*src, char \*oldVar, char \*newVar) { char buffer[100] = "";

char \*pos = src, \*match;

while ((match = strstr(pos, oldVar)) != NULL) { strncat(buffer, pos, match - pos); strcat(buffer, newVar);

pos = match + strlen(oldVar);

}

strcat(buffer, pos); strcpy(src, buffer);

}

int main() { FILE \*fp;

TAC tac[MAX];

char line[100], \*lhs, \*rhs; int count = 0;

// Open input file

fp = fopen("input.txt", "r"); if (!fp) {

printf("Error: Could not open 'input.txt'\n"); return 1;

}

// Read input file

while (fgets(line, sizeof(line), fp)) { line[strcspn(line, "\n")] = 0;

lhs = strtok(line, "="); rhs = strtok(NULL, "\n"); if (lhs && rhs) {

strcpy(tac[count].lhs, lhs); strcpy(tac[count].rhs, rhs); count++;

}

}

fclose(fp);

// Step 1: Common Subexpression Elimination (CSE) for (int i = 0; i < count; i++) {

for (int j = i + 1; j < count; j++) {

if (strcmp(tac[i].rhs, tac[j].rhs) == 0) {

replaceVar(tac[j + 1].rhs, tac[j].lhs, tac[i].lhs); strcpy(tac[j].lhs, "");

strcpy(tac[j].rhs, "");

}

}

}

// Step 2: Copy Propagation for (int i = 0; i < count; i++) {

if (strchr(tac[i].rhs, '+') == NULL && strchr(tac[i].rhs, '-') == NULL && strchr(tac[i].rhs, '\*') == NULL && strchr(tac[i].rhs, '/') == NULL) {

// rhs is a direct copy

for (int j = i + 1; j < count; j++) { replaceVar(tac[j].rhs, tac[i].lhs, tac[i].rhs);

}

// mark line as empty strcpy(tac[i].lhs, "");

strcpy(tac[i].rhs, "");

}

}

// Step 3: Dead Code Elimination for (int i = 0; i < count; i++) {

if (tac[i].lhs[0] != '\0' && !isUsed(tac, count, tac[i].lhs, i)) { strcpy(tac[i].lhs, "");

strcpy(tac[i].rhs, "");

}

}

// Print Optimized Code

printf("\nOptimized Code:\n \n");

for (int i = 0; i < count; i++) { if (tac[i].lhs[0] != '\0') {

printf("%s=%s\n", tac[i].lhs, tac[i].rhs);

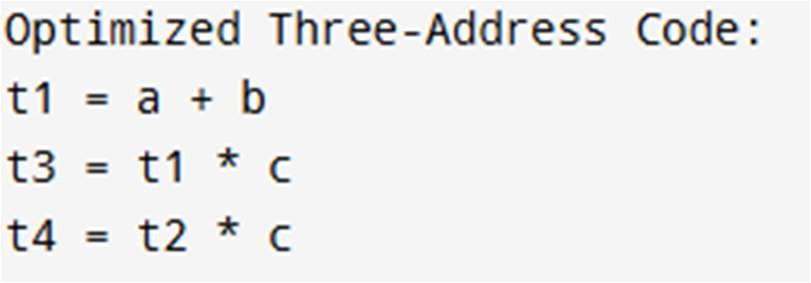
}

}

return 0;

}

**OUTPUT :**

****

|  |  |
| --- | --- |
| **Implementation** |  |
| **Output/Signature** |  |

**RESULT:**

Thus The Above Program To Implement Code Optimization Techniques Like Dead Code And Common Expression Elimination Is Executed And Implemented Successfully.