EXP NO : 10 DATE :

GENERATE THREE ADDRESS CODES FOR A GIVEN EXPRESSION

(ARITHMETIC EXPRESSION, FLOW OF CONTROL)

AIM:

The aim is to generate Three-Address Code (TAC) for a given arithmetic expression and flow of control (e.g., if-else, loops). TAC is an intermediate representation used in compilers to simplify the task of code generation. It consists of simple instructions that make it easier to translate into machine-level code.

For example, for an arithmetic expression a = b + c \* d, the TAC would break it down into simpler steps, using temporary variables to hold intermediate results.

ALGORITHM:

* The expression is read from the file using a file pointer
* Each string is read and the total no. of strings in the file is calculated.
* Each string is compared with an operator; if any operator is seen then the previous string and next string are concatenated and stored in a first temporary value and the three address code expression is printed
* Suppose if another operand is seen then the first temporary value is concatenated to the next string using the operator and the expression is printed.
* The final temporary value is replaced to the left operand value.

PROGRAM:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <ctype.h>

int tempCount = 1;

// Function to generate a temporary variable name

void newTemp(char \*temp) { sprintf(temp, "t%d", tempCount++);

}

// Function to generate TAC for arithmetic expressions like a = b + c \* d

void generateTACForExpression(char expr[]) { char lhs[20], rhs[100]; char op1[20], op2[20], result[20], operator; char temp1[10], temp2[10];

int i = 0, j = 0, len = strlen(expr);

// Split LHS and RHS

char \*equal = strchr(expr, '=');

if (!equal) { printf("; Invalid expression: %s\n", expr); return;

}

strncpy(lhs, expr, equal - expr); lhs[equal - expr] = '\0';

strcpy(rhs, equal + 1);

// Remove spaces char rhs\_clean[100]; for (int k = 0; rhs[k]; k++) { if (!isspace(rhs[k])) rhs\_clean[j++] = rhs[k];

}

rhs\_clean[j] = '\0';

// Handle binary operators: +, -, \*, /

// We'll scan from right to left to handle precedence (e.g., \* before +) char \*opPtr = NULL; if ((opPtr = strrchr(rhs\_clean, '\*')) || (opPtr = strrchr(rhs\_clean, '/')) ||

(opPtr = strrchr(rhs\_clean, '+')) ||

(opPtr = strrchr(rhs\_clean, '-'))) {

operator = \*opPtr; \*opPtr = '\0'; strcpy(op1, rhs\_clean);

strcpy(op2, opPtr + 1);

newTemp(temp1);

printf("%s = %s %c %s\n", temp1, op1, operator, op2); printf("%s = %s\n", lhs, temp1);

} else {

// Just direct assignment printf("%s = %s\n", lhs, rhs\_clean);

}

}

// Function to generate TAC for if/while (very simple form) void generateTACForControl(char line[]) { char cond[50], label1[10], label2[10]; static int labelCount = 1;

if (strstr(line, "if") != NULL) { sscanf(line, "if (%[^)])", cond); sprintf(label1, "L%d", labelCount++); sprintf(label2, "L%d", labelCount++); printf("if not %s goto %s\n", cond, label1); printf(" ; [true block statements]\n"); printf("goto %s\n", label2); printf("%s:\n", label1); printf(" ; [else block statements]\n"); printf("%s:\n", label2); } else if (strstr(line, "while") != NULL) { sscanf(line, "while (%[^)])", cond); sprintf(label1, "L%d", labelCount++); sprintf(label2, "L%d", labelCount++); printf("%s:\n", label1); printf("if not %s goto %s\n", cond, label2); printf(" ; [loop body statements]\n"); printf("goto %s\n", label1); printf("%s:\n", label2);

} else {

printf("; Unknown control statement: %s\n", line);

}

}

int main() { FILE \*fp;

char line[100];

fp = fopen("input.txt", "r"); if (fp == NULL) { printf("Error opening input.txt\n");

return 1;

}

printf("--- Three Address Code ---\n");

while (fgets(line, sizeof(line), fp)) { // Remove newline

line[strcspn(line, "\n")] = '\0';

if (strstr(line, "if") || strstr(line, "while")) {

generateTACForControl(line);

} else {

generateTACForExpression(line);

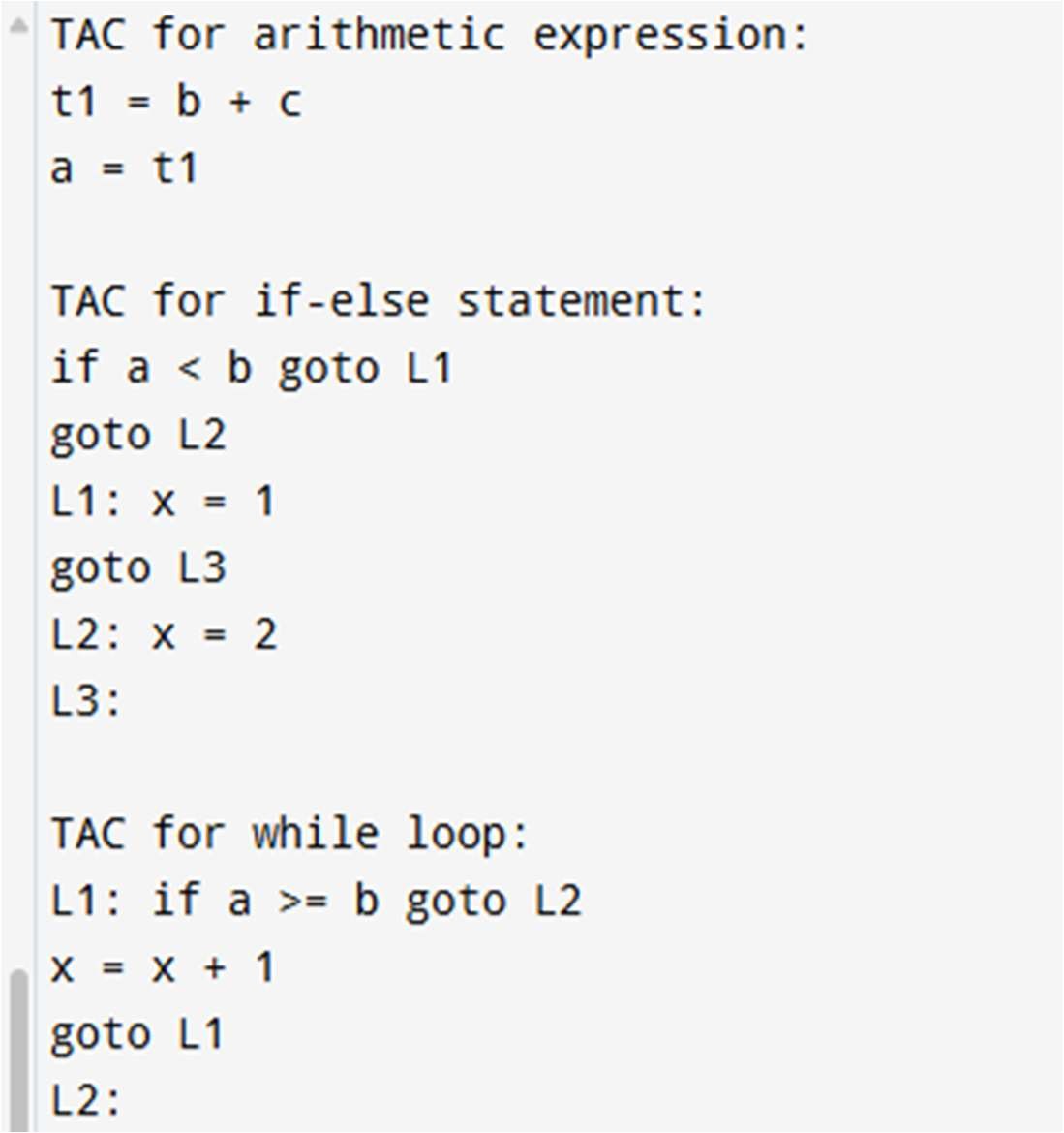
}

}

fclose(fp); return 0;

}

OUTPUT :



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

RESULT:

Thus the above program is the simplified example and a complete implementation and it would need to handle more complex expressions, nested control structures, and ensure proper parsing of the input.