CSE23CT302- THEORY OF COMPUTATION AND COMPILER DESIGN GitHub

Assignment 1. Implement a simple code generator that translates arithmetic expressions into target assembly for a stack machine.

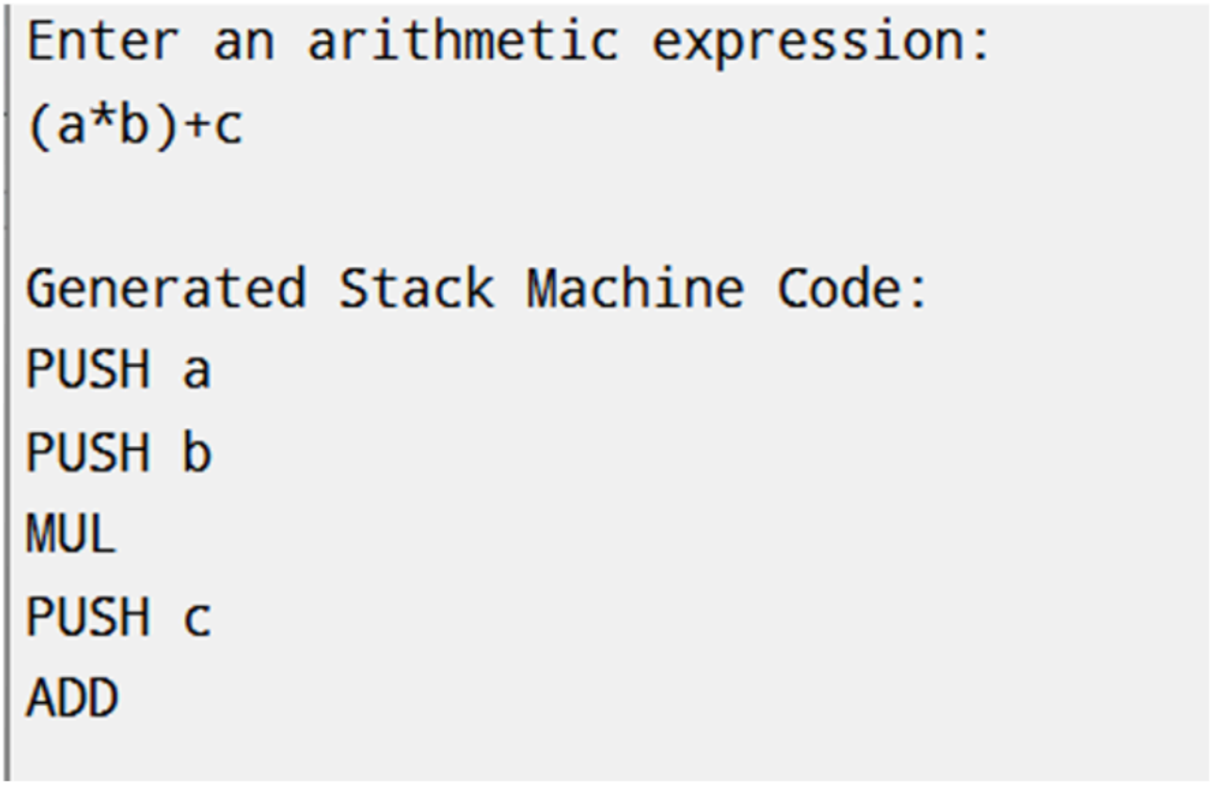
GitHub Link:https://github.com/abeejay13/Abeejay

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

#define MAX_TOKENS 100
#define MAX_TOKEN_LEN 32
#define MAX_CODE_LINES 100
#define MAX_EXPR_LEN 256

char tokens[MAX_TOKENS][MAX_TOKEN_LEN];
int tokenCount = 0;

char opStack[MAX_TOKENS][MAX_TOKEN_LEN];
int opTop = -1;

char postfix[MAX_TOKENS][MAX_TOKEN_LEN];
int postfixCount = 0;

char code[MAX_CODE_LINES][MAX_TOKEN_LEN];
int codeCount = 0;

void pushOp(const char *token) {
   strcpy(opStack[++opTop], token);
}

char* popOp() {
   return opStack[opTop--];
}

char* peekOp() {
   return opTop >= 0 ? opStack[opTop] : NULL;
```

```
}

int precedence(const char *op) {
    if (strcmp(op, "+") == 0 || strcmp(op, "-") == 0) return 1;
    if (strcmp(op, "*") == 0 || strcmp(op, "/") == 0) return 2;
    return 0;
}

int isOperator(const char *token) {
    return strcmp(token, "+") == 0 || strcmp(token, "-") == 0 ||
           strcmp(token, "*") == 0 || strcmp(token, "/") == 0;
}

void tokenize(const char *expr) {
    tokenCount = 0;
    int i = 0;
    while (expr[i]) {
        if (isspace(expr[i])) {
            i++;
            continue;
        }

        if (isalnum(expr[i]) || expr[i] == '_') {
            int j = 0;
            while (isalnum(expr[i]) || expr[i] == '_') {
                tokens[tokenCount][j++] = expr[i++];
            }
            tokens[tokenCount][j] = '\0';
            tokenCount++;
        } else if (strchr("+-*/()", expr[i])) {
            tokens[tokenCount][0] = expr[i++];
            tokens[tokenCount][1] = '\0';
            tokenCount++;
        } else {
            i++;
        }
    }
}

void infixToPostfix() {
    postfixCount = 0;
    opTop = -1;

    for (int i = 0; i < tokenCount; i++) {
        char *token = tokens[i];

        if (isalnum(token[0]) || token[0] == '_') {
            strcpy(postfix[postfixCount++], token);
        } else if (isOperator(token)) {
            while (opTop >= 0 && strcmp(peekOp(), "(") != 0 &&
                   precedence(peekOp()) >= precedence(token)) {
                strcpy(postfix[postfixCount++], popOp());
            }
            pushOp(token);
        } else if (strcmp(token, "(") == 0) {
            pushOp(token);
        } else if (strcmp(token, ")") == 0) {
            while (opTop >= 0 && strcmp(peekOp(), "(") != 0) {
                strcpy(postfix[postfixCount++], popOp());
            }
            if (opTop >= 0 && strcmp(peekOp(), "(") == 0) {
                popOp(); // discard '('
            }
        }
    }

    while (opTop >= 0) {
        strcpy(postfix[postfixCount++], popOp());
    }
}

void generateCode() {
    codeCount = 0;
    for (int i = 0; i < postfixCount; i++) {
        char *token = postfix[i];
        if (isOperator(token)) {
            if (strcmp(token, "+") == 0) strcpy(code[codeCount++], "ADD");
            else if (strcmp(token, "-") == 0) strcpy(code[codeCount++], "SUB");
            else if (strcmp(token, "*") == 0) strcpy(code[codeCount++], "MUL");
            else if (strcmp(token, "/") == 0) strcpy(code[codeCount++], "DIV");
        } else {
            char line[MAX_TOKEN_LEN + 6];
            snprintf(line, sizeof(line), "PUSH %s", token);
            strcpy(code[codeCount++], line);
        }
    }
}
```

```
int main() {
    char expr[MAX_EXPR_LEN];

    printf("Enter an arithmetic expression:\n");
    fgets(expr, sizeof(expr), stdin);
    expr[strcspn(expr, "\n")] = 0;

    tokenize(expr);
    infixToPostfix();
    generateCode();

    printf("\nGenerated Stack Machine Code:\n");
    for (int i = 0; i < codeCount; i++) {
        printf("%s\n", code[i]);
    }

    return 0;
}
```

Output:



```
Enter an arithmetic expression:
(a*b)+c

Generated Stack Machine Code:
PUSH a
PUSH b
MUL
PUSH c
ADD
```