Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра вычислительных методов и программирования

Отчет по лабораторной работе №4
Обратная польская запись
Вариант 11

Выполнил:
студент 1 курса
группы № 348602
Трошкин Дмитрий Сергеевич

Проверил:
Матюшкин Светослав Иванович

Минск 2023

1 ОБРАТНАЯ ПОЛЬСКАЯ ЗАПИСЬ

Цель работы: изучить правила формирования постфиксной записи арифметических выражений с использованием стека.

1.1 Условие

Написать программу формирования ОПЗ и расчета полученного выражения. Разработать удобный интерфейс ввода исходных данных и вывода результатов. Работу программы проверить на конкретном примере (табл. 4.1).

1.2 Исходный код

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define LETTERS_COUNT ('z' - 'a' + 1)
#define OPERATIONS "+-*/^"
#define MAXSIZE 1024

typedef struct Stack {
    char data;
    struct Stack *next;
} stack;

typedef struct Manipulation { // Действие
    char operation; // stores operation or variable name
    struct Manipulation *a, *b;
} manip;

double values[LETTERS_COUNT * 2];

int isUpper(char c) { return ('A' <= c) && ('Z' >= c); }
int isLover(char c) { return ('a' <= c) && ('z' >= c); }
int isLatin(char c) { return isUpper(c) || isLover(c);}
int isNum(char c) { return (c >= '0') && (c <= '9'); }

void push(stack **s, char val)
{
        stack *t = malloc(sizeof(stack));
        t->next = *s;
        t->data = val;
        *s = t;
}

int pop(stack **s, char *val)
```

```
{
        stack *t;
        if(*s == NULL) return -1;
        *val = (*s)->data;
        t = *s;
        *s = t->next;
        free(t);
        return 0;
}

int lenstr(char *str) {
    int i = 0;
    for(; str[i]; i++);
    return i;
}

int getIndex(const char *str, char c) {
    for(int i = 0; str[i]; i++)
        if(str[i] == c) return i;
    return -1;
}

int isOperation(char c) { return getIndex(OPERATIONS, c) != -1; }

void clean_value()
{
    for(int i = 0; i < LETTERS_COUNT * 2; i++)
        values[i] = 0;
}
void save_value(char name, double val)
{
    if(isLover(name)) values[name - 'a'] = val;
    if(isUpper(name)) values[name - 'A' + LETTERS_COUNT] = val;
}
double get_value(char name)
{
    if(isLover(name)) return values[name - 'a'];
    if(isUpper(name)) return values[name - 'A' + LETTERS_COUNT];
    return atoi(&name);
}
void read_values()
{
    for(int i = 0; i < LETTERS_COUNT * 2; i++) {
        if(!values[i]) continue;
        printf("Enter \'%c\' value: ",
               i + (i > LETTERS_COUNT ? 'A' - LETTERS_COUNT: 'a'));
        scanf("%lf", values+i);
    }
}

void initManipulation(manip **m, char operation)
```

```
{
    *m = malloc(sizeof(manip));
    (*m)->operation = operation;
    (*m)->a = (*m)->b = NULL;
}

void viewManipTree(manip *m)
{
    if(m) {
        viewManipTree(m->a);
        viewManipTree(m->b);
        printf("%c", m->operation);
    }
}

void clearManipTree(manip *m)
{
    if(m) {
        clearManipTree(m->a);
        clearManipTree(m->b);
        free(m);
    }
}

int isCoreectInfix(char *infix)
{
    int isOper = 1;
    int openScobka = 0;
    for(;*infix;infix++)
    {
        if(*infix == '(') openScobka++;
        else if (isLatin(*infix) || isNum(*infix)) {
            if(isOper == 0) return 0;
            isOper = 0;
        } else if(isOperation(*infix)) {
            if(isOper == 1) return 0;
            isOper = 1;
        } else if(*infix == ')') {
            if(isOper == 1) return 0;
            openScobka--;
        }
    }
    return !openScobka && !isOper;
}

int reversePolishNotationToManipTree(manip **m, char *str, int size)
{
    int dsize = 1;
    if(size > 0) {
        *m = malloc(sizeof(manip));
        (*m)->operation = str[size-1];
```

```
        if(!isLatin((*m)->operation) && !isNum((*m)->operation)) {
            dsize += reversePolishNotationToManipTree(&(*m)->b, str, size - dsize);
            dsize += reversePolishNotationToManipTree(&(*m)->a, str, size - dsize);
        }
    }
    return dsize;
}

int priority(char c)
{
    switch(c){
        case '^':           return 4;
        case '*': case '/': return 3;
        case '+': case '-': return 2;
        case '(':           return 1;
    }
    return 0;
}

void infixToRPN(char **output, char *str)
{
    char a;
    int length, index = 0;
    stack *begin = NULL;
    length = lenstr(str);
    *output = malloc(length * sizeof(char));

    for(;*str; str++) {
        if(*str == '(') push(&begin, *str);
        if(*str == ')') {
            while(begin->data != '(')
                pop(&begin, *output + index++);
            pop(&begin, &a);
        }
        if(isLatin(*str)) {
            (*output)[index++] = *str;
            save_value(*str, -1);
        }
        if(isNum(*str))
            (*output)[index++] = *str;
        if(isOperation(*str)) {
            while(begin && priority(begin->data) >= priority(*str))
                pop(&begin, *output + index++);
            push(&begin, *str);
        }
    }
    while(begin)
        pop(&begin, *output + index++);
}

double solve(manip *m)
```

```
{
    switch(m->operation) {
        case '+': return solve(m->a) + solve(m->b);
        case '-': return solve(m->a) - solve(m->b);
        case '*': return solve(m->a) * solve(m->b);
        case '/': return solve(m->a) / solve(m->b);
        case '^': return pow(solve(m->a), solve(m->b));
        default: return get_value(m->operation);
    }
}

int main()
{
    char *rpn, *infix = malloc(MAXSIZE * sizeof(char));
    manip *m = NULL;

    printf("Enter infix expression: ");
    scanf("%s", infix);

    if(isCoreectInfix(infix)) {
        clean_value();
        infixToRPN(&rpn, infix);
        read_values();
        printf("%s\n", rpn);
        reversePolishNotationToManipTree(&m, rpn, lenstr(rpn));
        viewManipTree(m);

        double t = solve(m);
        printf("\n%lf\n", t);
        clearManipTree(m);
        free(infix);
    } else
        printf("The following string isn't correct infix expression:\n%s", infix);
}
```

1.3 Пример

```
 $ ./task
Enter infix expression: a-(b/c*(d+e))
Enter 'a' value: 5.6
Enter 'b' value: 3.2
Enter 'c' value: 0.9
Enter 'd' value: 1.7
Enter 'e' value: 4.8
abc/de+*-
abc/de+*-
-17.511111
```