Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

—

Институт компьютерных наук и кибербезопасности

**Высшая школа кибербезопасности**

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**Калькулятор**

по дисциплине «Структуры данных»

Выполнили

студенты гр. 5131001/30002 Мишенев Н. С.

Квашенникова В. М.

<*подпись*>

Руководитель

программист Вагисаров В. Б.

<*подпись*>

Санкт-Петербург

2024г.

СОДЕРЖАНИЕ

[ЦЕЛЬ РАБОТЫ 3](#__RefHeading___1)

[ХОД РАБОТЫ 4](#__RefHeading___2)

[ВЫВОД 6](#__RefHeading___3)

[ПРИЛОЖЕНИЕ А 7](#__RefHeading___4)

ЦЕЛЬ РАБОТЫ

Написать программу, осуществляющую подсчет выражения на основе алгоритма подсчета значения в постфиксной нотации и алгоритма сортировочной станции с использованием структур данных стек и очередь.

ХОД РАБОТЫ

1. **Реализация структур данных стек и очередь**

Структуры данных стек и очередь необходимо было реализовать на основе линейных списков. Таким образом, сначала была описана структура одного узла линейного списка, а затем уже добавлены описания стека и очереди. Все описания были вынесены в заголовочный файл “obj.h”, в нем хранятся и основные функции для взаимодействия с введенными структурами.

1. **Функции для работы со стеком и очередью**

Были реализованы следующие функции для работы со стеком:

1. void initializeStack(struct Stack\* s);
2. int isEmptyS(struct Stack\* s);
3. void printStack(struct Stack\* s);
4. void push(struct Stack\* s, char\* data);
5. void pop(struct Stack\* s, char\* result);

Также функции для работы с очередью:

1. void initializeQueue(struct Queue\* q);
2. int isEmptyQ(struct Queue\* q);
3. void put(struct Queue\* q, char\* data);
4. void get(struct Queue\* q, char\* result);
5. void freeQueue(struct Queue\* q);
6. void printQueue(struct Queue\* q)
7. **Перевод инфиксной нотации в постфиксную**

Весь алгоритм сортировочной станции вынесен в функцию struct Queue\* postfix\_notation(char\* buffer), которая возвращает указатель на очередь, в которой хранится выражение в постфиксной нотации. Функция работает по алгоритму перегонки вагонов или по-другому алгоритму Дейкстры.

Принцип работы алгоритма Дейкстры:

1. Посимвольно анализируем исходную строку;
2. Если найдено число, оно заносится его в выходную строку;
3. При нахождении оператора, он заносится в стек;
4. Все операторы выталкиваются в выходную строку из стека, имеющие приоритет выше рассматриваемого;
5. При нахождении открывающейся скобки, она заносится её в стек;
6. При нахождении закрывающей скобки, выталкиваются из стека все операторы до открывающейся скобки, а открывающуюся скобку удаляем из стека.

После выполнения алгоритма функция возвращает указатель на очередь, которая затем участвует в вычислении значения выражения.

1. **Функция вычисления**

Принцип работы алгоритма вычисления постфиксной нотации заключается в следующем:

1. Выполняется обход постфиксной записи.
2. При обнаружении числа оно помещается в стек.
3. При обнаружении бинарного оператора из стека извлекаются два последних значения в обратном порядке.
4. При обнаружении унарного оператора (в данном случае унарного минуса) из стека извлекается последнее значение и вычитается из нуля, поскольку унарный минус является правосторонним оператором.
5. Последнее значение, оставшееся в стеке после выполнения всех операций, является результатом выражения.

ВЫВОД

Была реализована программа, осуществляющая подсчет выражения на основе алгоритма подсчета значения в постфиксной нотации и алгоритма Дейкстры с использованием структур данных стек и очередь.

ПРИЛОЖЕНИЕ А

Листинг 1 - исходный код программы на языке С

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "obj.h"

#include "functions.h"

#define BUFFERLENGTH 400

#define ALPLENGTH 25

struct Queue\* postfix\_notation(char\* buffer);

void calculator(struct Queue\* q, char\* operands, char values[][BUFFERSIZE], int per\_amt);

int check\_operands(char\* operands, char\* buffer);

//Converts an expression in infix notation to postfix one using Dijkstra's algorithm.

struct Queue\* postfix\_notation(char\* buffer)

{

struct Stack Tehas;

initializeStack(&Tehas);

struct Queue California;

initializeQueue(&California);

char tempstr[BUFFERSIZE];

Clear(tempstr, BUFFERSIZE);

for (int i = 0; buffer[i] != '\0'; i++)

{

if (isalpha(buffer[i]))

{

int j = 0;

while (isalpha(buffer[i])) tempstr[j++] = buffer[i++];

i--;

if (strlen(tempstr) == 1) //if variables put in queue

{

put(&California, tempstr);

}

else goto operationslogic;//if functions

Clear(tempstr, BUFFERSIZE);

}

else if (isdigit(buffer[i]))

{

int j = 0;

while (isdigit(buffer[i]) || buffer[i] == '.') tempstr[j++] = buffer[i++];

i--;

put(&California, tempstr);

Clear(tempstr, BUFFERSIZE);

}

else if (buffer[i] == '(')

{

tempstr[0] = buffer[i];

tempstr[1] = '\0';

push(&Tehas, tempstr);

Clear(tempstr, BUFFERSIZE);

}

else if (buffer[i] == ')')

{

char r[BUFFERSIZE];

while (!isEmptyS(&Tehas) && Tehas.top->data[0] != '(') {

char result[BUFFERSIZE] = { 0 };

pop(&Tehas, result);

put(&California, result);

}

if (!isEmptyS(&Tehas)) pop(&Tehas, r);

}

else if (buffer[i] != ' ') //operations like + - \* etc.

{

tempstr[0] = buffer[i];

tempstr[1] = '\0';

operationslogic://!!! here functions like sqrt sin cos and etc. are added.

while (!isEmptyS(&Tehas) && operation\_priority(Tehas.top->data) >= operation\_priority(tempstr)) {

char result[BUFFERSIZE] = { 0 };

pop(&Tehas, result);

put(&California, result);

}

push(&Tehas, tempstr);

Clear(tempstr, BUFFERSIZE);

}

}

while (!isEmptyS(&Tehas)) {

char result[BUFFERSIZE] = { 0 };

pop(&Tehas, result);

put(&California, result);

}

return &California;

}

//Doing some operations in postfix notatin in queue with stack

void calculator(struct Queue\* q, char\* operands, char values[][BUFFERSIZE], int per\_amt)

{

struct list\* current = q->first;

struct Stack stack;

initializeStack(&stack);

char res[BUFFERSIZE] = { 0 };

while (current != NULL)

{

Clear(res, BUFFERSIZE);

get(q, res);

int flag = demarcation(res);

switch (flag)

{

case 1:

push(&stack, res);

break;

case 2:

{

int ind = massivesearch(operands, res[0], per\_amt);

if (ind != (-1)) push(&stack, values[ind]);

break;}

case 3:

calculation(&res, &stack);

break;

default:

break;

}

current = current->pnextList;

}

printf("\nRESULT:%s", stack.top->data);

}

//Add variable names from buffer in operands massive.

//Return variebles amount

int check\_operands(char\* operands, char\* buffer)

{

int kol = 0;

int flag = 0;

for (int i = 0; buffer[i] != '\0'; i++)

{

flag = 0;

if (isalpha(buffer[i]))

{

if ((i == 0 && !isalpha(buffer[i + 1])) || (i > 0 && buffer[i + 1] != '\0' && !isalpha(buffer[i + 1]) && !isalpha(buffer[i - 1])) || (i > 0 && buffer[i + 1] == '\0' && !isalpha(buffer[i - 1])))

{

flag = 1;

}

if (flag == 1)

{

if (massivesearch(operands, buffer[i], kol) == (-1)) {

operands[kol++] = buffer[i];

}

}

}

}

return kol;

}

int main(void)

{

char buffer[BUFFERLENGTH] = { 0 };

char operands[ALPLENGTH] = { 0 };

printf("Enter expression>>>> ");

fflush(stdin);

fgets(buffer, 400, stdin);

buffer[strlen(buffer) - 1] = '\0';

int per\_amt = check\_operands(operands, buffer);

char values[ALPLENGTH][BUFFERSIZE];

for (int i = 0; i < per\_amt; i++)

{

printf("Enter value of variable %c >>>", operands[i]);

fflush(stdin);

fgets(&values[i], BUFFERSIZE, stdin);

values[i][strlen(values[i]) - 1] = '\0';

}

struct Queue\* q = postfix\_notation(buffer);

printf("Expression in postfix notation: ");

printQueue(q);

printf("\n");

calculator(q, operands, values, per\_amt);

}

Листинг 2 – исходный код заголовочного файла с реализацией структур данных стек и очередь на языке С

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#include <stdio.h>

#define BUFFERSIZE 20

void Clear(char\* str, int strlen);

struct list;

struct Stack;

struct Queue;

//stack:

void initializeStack(struct Stack\* s);

int isEmptyS(struct Stack\* s);

void printStack(struct Stack\* s);

void push(struct Stack\* s, char\* data);

void pop(struct Stack\* s, char\* result);

//queue:

void initializeQueue(struct Queue\* q);

int isEmptyQ(struct Queue\* q);

void put(struct Queue\* q, char\* data);

void get(struct Queue\* q, char\* result);

void freeQueue(struct Queue\* q);

void printQueue(struct Queue\* q);

void Clear(char\* str, int strlen) {

for (int i = 0; i < strlen; i++) {

str[i] = 0;

}

}

struct list

{

struct list\* pnextList;

char data[BUFFERSIZE];

};

struct Stack{

struct list \*top;

};

void initializeStack(struct Stack \*s) {

s->top = NULL;

}

int isEmptyS(struct Stack \*s) {

return (s->top == NULL);

}

void printStack(struct Stack \*s) {

if (isEmptyS(s)) {

printf("Stack is empty.\n");

return;

}

printf("Stack contents:");

struct list \*current = s->top;

while (current != NULL) {

printf("!%s! ", current->data);

current = current->pnextList;

}

printf("\n");

}

void push(struct Stack\* s, char\* data) {

struct list\* newlist = (struct list\*)malloc(sizeof(struct list));

if (newlist == NULL) {

printf("Error: Memory allocation failed.\n");

return;

}

Clear(newlist->data, BUFFERSIZE);

newlist->pnextList = s->top;

s->top = newlist;

strncpy(newlist->data, data, BUFFERSIZE - 1);

newlist->data[BUFFERSIZE - 1] = '\0'; // Гарантируем нуль-терминацию

}

void pop(struct Stack\* s, char\* result) {

if (isEmptyS(s)) {

printf("Error: Stack is empty\n");

return;

}

struct list\* temp = s->top;

s->top = temp->pnextList;

if (result != NULL && temp->data != NULL) {

strcpy(result, temp->data);

}

free(temp);

}

struct Queue{

struct list\* first;

struct list\* last;

};

void initializeQueue(struct Queue \*q) {

q->first = NULL;

q->last = NULL;

}

int isEmptyQ(struct Queue \*q) {

return (q->first == NULL);

}

void put(struct Queue \*q,char\* data) {

struct list \*newlist = (struct list \*)malloc(sizeof(struct list));

if (newlist == NULL) {

printf("Memory allocation failed.\n");

return;

}

strcpy(newlist->data,data);

newlist->pnextList = NULL;

// If the queue is empty, the new node becomes the start and end of the queue

if (isEmptyQ(q)) {

q->first = newlist;

q->last = newlist;

} else {

q->last->pnextList = newlist;

q->last = newlist;

}

}

//first el of queue

void get(struct Queue\* q, char\* result) {

if (isEmptyQ(q)) {

printf("Queue is empty. Unable to get.\n");

return;

}

struct list\* temp = q->first;

if (temp->data != NULL && result != NULL) {

strcpy(result, temp->data);

// printf("copied %s\n",result);

}

else {

printf("Error: Invalid pointers detected.\n");

return;

}

if (q->first == NULL) {

q->last = NULL;

}

q->first = q->first->pnextList;

//free(temp);

}

// Освобождение памяти, выделенной под элементы очереди

void freeQueue(struct Queue \*q) {

while (!isEmptyQ(q)) {

get(q,NULL);

}

}

void printQueue(struct Queue \*q) {

if (isEmptyQ(q)) {

printf("Queue is empty.\n");

return;

}

// printf("Queue contents: ");

struct list \*current = q->first;

while (current != NULL) {

printf("%s ", current->data);

current = current->pnextList;

}

}

Листинг 3 - исходный заголовочного файла с вспомогательными функциями программы на языке C

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <math.h>

#define M\_PI 3.14159265358979323846

double factorial(double n);

void calculation(char\* operation, struct Stack\* s);

int massivesearch(char massive[], char data, int n);

int operation\_priority(char\* operation);

int demarcation(char\* str);

double factorial(double n) {

if (n < 0) {

printf("Error: negative value\n");

return -1.0;

}

double result = 1.0;

for (int i = 1; i <= (int)n; i++) {

result \*= i;

}

return result;

}

void calculation(char\* operation, struct Stack\* s) {

char operations[15][5] = { "cos" , "sin" , "tg" , "ctg" , "asin", "acos" , "atg", "actg", "sqrt", "!", "^", "/", "\*", "+", "-" };

char value1[BUFFERSIZE] = { 0 }, value2[BUFFERSIZE] = { 0 };

if (strcmp(operations[0], operation) == 0) {

pop(s, value1);

double result = cos(atof(value1));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[1], operation) == 0) {

pop(s, value1);

double result = sin(atof(value1));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[2], operation) == 0) {

pop(s, value1);

double result = tan(atof(value1));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[3], operation) == 0) {

pop(s, value1);

double result = tan(M\_PI / 2 - atof(value1));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[4], operation) == 0) {

pop(s, value1);

double result = acos(atof(value1));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[5], operation) == 0) {

pop(s, value1);

double result = asin(atof(value1));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[6], operation) == 0) {

pop(s, value1);

double result = atan(atof(value1));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[7], operation) == 0) {

pop(s, value1);

double result = M\_PI / 2 - atan(atof(value1));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[8], operation) == 0) {

pop(s, value1);

if (atof(value1) < 0) printf("\nERROR: value in sqrt is negative");

double result = sqrt(atof(value1));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[9], operation) == 0) {

pop(s, value1);

if (atof(value1) < 0) printf("\nERROR: value in factorial is negative");

double result = factorial(atof(value1));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[10], operation) == 0) {

pop(s, value2);

pop(s, value1);

double result = pow(atof(value1), atof(value2));

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[11], operation) == 0) {

pop(s, value2);

pop(s, value1);

double result = atof(value1) / atof(value2);

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[12], operation) == 0) {

pop(s, value2);

pop(s, value1);

double result = atof(value1) \* atof(value2);

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[13], operation) == 0) {

pop(s, value2);

pop(s, value1);

double result = atof(value1) + atof(value2);

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

else if (strcmp(operations[14], operation) == 0) {

pop(s, value2);

pop(s, value1);

double result = atof(value1) - atof(value2);

char resultStr[BUFFERSIZE];

snprintf(resultStr, BUFFERSIZE, "%f", result);

push(s, resultStr);

}

}

int massivesearch(char massive[], char data, int n)

{

for (int i = 0; i < n; i++)

{

if (massive[i] == data) return i;

}

return -1;

}

//Get operation priority

int operation\_priority(char\* operation)

{

if (strlen(operation) == 1)

{

if (operation[0] == '^' || operation[0] == '!') return 3;

if (operation[0] == '/' || operation[0] == '\*') return 2;

if (operation[0] == '+' || operation[0] == '-') return 1;

if (operation[0] == '(') return 0;

}

else if (strlen(operation) > 1)

{

return 4;//functions sin cos etc.

}

else return -1;//unknown operation

}

//return 1 if digit

//return 2 if variable

//return 3 if operation or function

int demarcation(char\* str)

{

if (isdigit(str[0])) return 1;

else if (isalpha(str[0]) && strlen(str) == 1) return 2;

else if (strlen(str) == 0) return -1; //some trash

else return 3;//operations and functions

}