Министерство науки и высшего образования РФ

ФГАОУ ВО Пермский национальный исследовательский

политехнический университет

Кафедра «Вычислительная математика, механика и биомеханика»

Отчет по лабораторной работе № 2

тема «Функции, массивы, структуры и рекурсия в языке программирования C»

по дисциплине «Языки программирования и методы трансляции»

Выполнил: студент группы ИСТ-22-1Б Синьковский Г.С.

Проверил: Батин С.Е.

Пермь, 2022

Содержимое

[Упражнение 1 3](#_Toc149772504)

[1.1. Постановка задачи 3](#_Toc149772505)

[1.2. Код программы 3](#_Toc149772506)

[1.3 Пример работы программы 4](#_Toc149772507)

[Упражнение 2 6](#_Toc149772508)

[2.1. Постановка задачи 6](#_Toc149772509)

[2.2. Код программы 6](#_Toc149772510)

[2.3. Примеры работы программы 7](#_Toc149772511)

[Упражнение 3 10](#_Toc149772512)

[3.1. Постановка задачи 10](#_Toc149772513)

[3.2 Код программы 10](#_Toc149772514)

[3.3. Пример работы программы 12](#_Toc149772515)

[Упражнение 4 14](#_Toc149772516)

[4.1. Постановка задачи 14](#_Toc149772517)

[4.2. Код программы 14](#_Toc149772518)

[4.3. Пример работы программы 16](#_Toc149772519)

[Упражнение 5 16](#_Toc149772520)

[5.1 Постановка задачи 16](#_Toc149772521)

[5.2 Код программы 16](#_Toc149772522)

[5.3 Пример работы программы 19](#_Toc149772523)

[Упражнение 6 22](#_Toc149772524)

[6.1 Постановка задачи 22](#_Toc149772525)

[6.2 Код программы 22](#_Toc149772526)

[6.3 Пример работы программы 28](#_Toc149772527)

# Упражнение 1

## 1.1. Постановка задачи

Написать программу, которая генерирует массив из псевдослучайных целых чисел и выводит его в консоль в прямом порядке, и записывает в файл в обратном порядке.

## 1.2. Код программы

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#include <malloc.h>

int main()

{

// Инициализация генератора случайных чисел с использованием текущего времени

srand(time(NULL));

// Объявление переменной для длины массива

int len;

// Запрос пользователя на ввод длины массива

printf("Введите длину массива: ");

scanf("%d", &len);

// Выделение динамической памяти под массив

int \*arr = (int \*)malloc(len \* sizeof(int));

// Заполнение массива случайными числами и их вывод

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

{

arr[i] = 10 + rand() % (99 - 10 + 1);

printf("%d ", arr[i]);

}

// Открытие файла "exc1-1.txt" для записи

FILE \*file = fopen("exc1-1.txt", "w");

// Проверка успешного открытия файла

if (file == NULL)

exit(1);

// Запись элементов массива в обратном порядке в файл

for (int i = len - 1; i >= 0; i--)

{

fprintf(file, "%d ", arr[i]);

}

// Освобождение выделенной памяти для массива

free(arr);

// Закрытие файла

fclose(file);

// Возвращение значения 0, что указывает на успешное завершение программы

return 0;

}

## 1.3 Пример работы программы

На рис. 1, 2 продемонстрирован результат работы программы 1 упражнения.

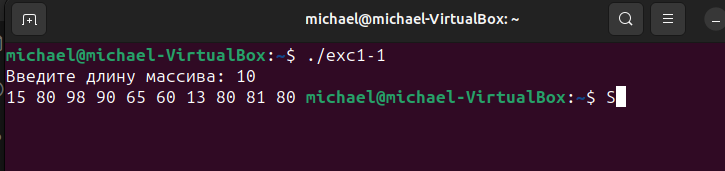


Рисунок 1. Результат программы 1 упражнения

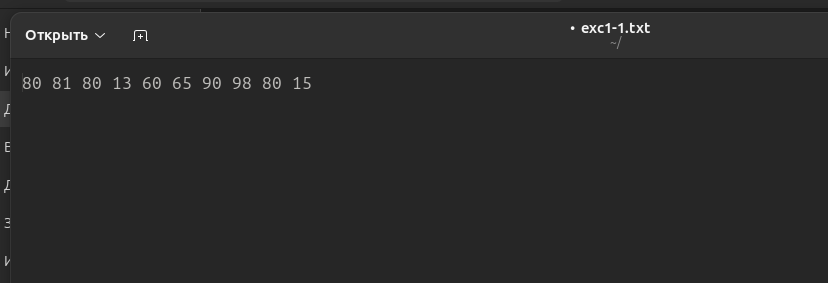


Рисунок 2. Результат программы 1 упражнения

# Упражнение 2

## 2.1. Постановка задачи

Написать программу, которая принимает размер произвольного массива, создает его, заполняет случайными числами с плавающей точкой в интервале (0, 1). Выводит этот массив в консоль и записывает в файл в отсортированном виде.

## 2.2. Код программы

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <malloc.h>

// Объявление прототипа функции сортировки

void sort(double \*arr, const int SIZE);

int main()

{

// Инициализация генератора случайных чисел с использованием текущего времени

srand(time(NULL));

// Объявление переменной для длины массива

int len;

// Запрос пользователя на ввод длины массива

printf("Input array length here: ");

scanf("%d", &len);

// Выделение динамической памяти под массив вещественных чисел

double \*array = malloc(len \* sizeof(double));

// Заполнение массива случайными вещественными числами от 0 до 1 и их вывод

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

{

double rand\_num = (0 + rand() % (1000 - 0 + 1)) / 1000.0;

\*(array + i) = rand\_num;

printf("%f ", rand\_num);

}

// Открытие файла "exc1-2.txt" для записи

FILE \*file = fopen("exc1-2.txt", "w");

// Проверка успешного открытия файла

if (file == NULL)

{

printf("Press F");

exit(1);

}

// Вызов функции сортировки массива

sort(array, len);

// Запись отсортированных элементов массива в файл

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

{

fprintf(file, "%f ", \*(array + i));

}

// Освобождение выделенной памяти для массива

free(array);

// Возвращение значения 0, что указывает на успешное завершение программы

return 0;

}

// Определение функции сортировки пузырьком для массива вещественных чисел

void sort(double \*arr, const int SIZE)

{

for (int i = 0; i < SIZE - 1; i++)

{

for (int j = 0; j < SIZE - i - 1; j++)

{

// Поменять местами элементы, если они стоят в неправильном порядке

if (\*(arr + j) > \*(arr + j + 1))

{

double temp = \*(arr + j);

\*(arr + j) = \*(arr + j + 1);

\*(arr + j + 1) = temp;

}

}

}

}

## 2.3. Примеры работы программы

На рис.3,4 показаны результаты работы программы упражнения 2.

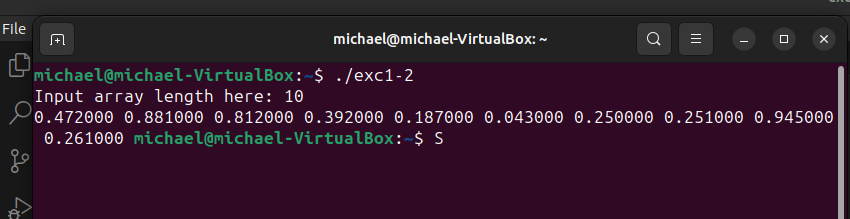
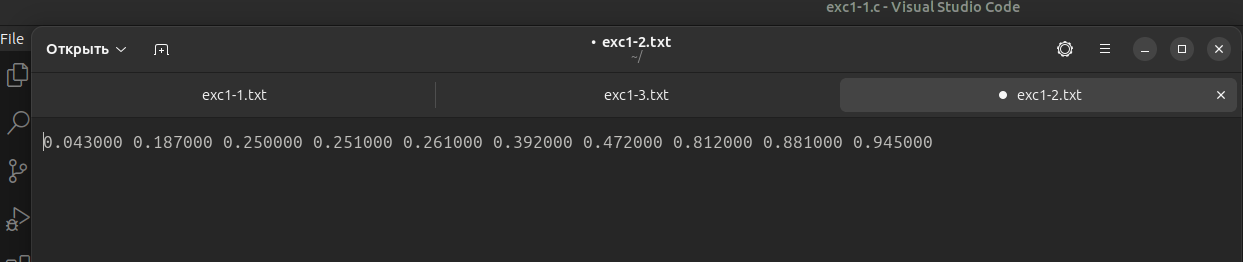


Рисунок 3. Результат работы программы упражнения 2

# Упражнение 3

## 3.1. Постановка задачи

Создать структуру для хранения данных о названии и цене товара. Считать из входного файла данные о товарах. Написать функцию для сортировки товаров по цене. Вывести отсортированные данные в выходной файл.

## 3.2 Код программы

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

// Структура, представляющая товар

struct product

{

char name[256];

int price;

};

// Прототип функции сортировки массива товаров по цене

void sort(struct product \*arr, const int SIZE);

// Функция для определения размера файла (количество строк)

int getFileSize(FILE \*file);

int main()

{

// Открытие файла "exc1-3.txt" для чтения

FILE \*file = fopen("exc1-3.txt", "r");

if (file == NULL)

{

// Вывод сообщения об ошибке, если файл не найден

printf("Error, file was not found");

exit(EXIT\_FAILURE);

}

// Определение размера файла (количество строк)

int fileSize = getFileSize(file);

// Выделение динамической памяти под массив товаров

struct product \*products = malloc(fileSize \* sizeof(struct product));

// Заполнение массива данными из файла и их вывод

int i = 0;

while (fscanf(file, "%s %d", products[i].name, &products[i].price) == 2)

{

fseek(file, 2, SEEK\_CUR);

printf("Name: %s; Price: %dр\n", products[i].name, products[i].price);

i++;

}

// Закрытие файла

fclose(file);

// Открытие файла "exc1-3Output.txt" для записи

file = fopen("exc1-3Output.txt", "w");

// Сортировка массива товаров по цене

sort(products, fileSize);

// Запись отсортированных товаров в файл

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

fprintf(file, "Name: %s; Cost: %dp\n", products[i].name, products[i].price);

// Закрытие файла

fclose(file);

// Освобождение выделенной памяти для массива товаров

free(products);

// Возвращение значения 0, что указывает на успешное завершение программы

return 0;

}

// Функция для определения размера файла (количество строк)

int getFileSize(FILE \*file)

{

int lineCount = 0;

char ch;

while ((ch = fgetc(file)) != EOF)

{

if (ch == '\n')

{

lineCount++;

}

}

lineCount--;

fseek(file, 0, SEEK\_SET);

return lineCount;

}

// Функция сортировки массива товаров по цене (пузырьковая сортировка)

void sort(struct product \*arr, const int SIZE)

{

for (int i = 0; i < SIZE - 1; i++)

{

for (int j = 0; j < SIZE - i - 1; j++)

{

// Поменять местами товары, если их цены стоят в неправильном порядке

if ((arr + j)->price > (arr + j + 1)->price)

{

struct product temp = \*(arr + j);

\*(arr + j) = \*(arr + j + 1);

\*(arr + j + 1) = temp;

}

}

}

}

## 3.3. Пример работы программы

Содержимое текстового файла test.txt (рис.4):

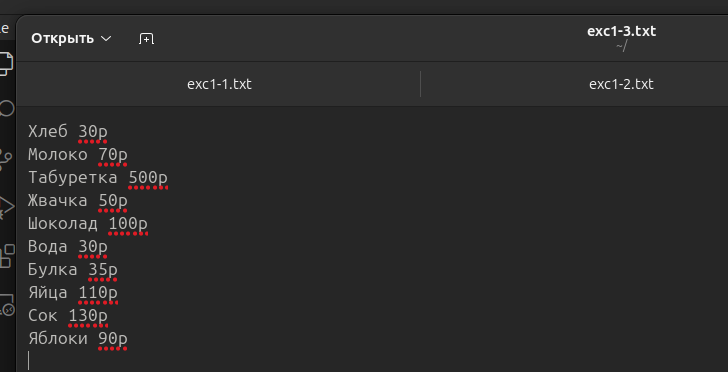
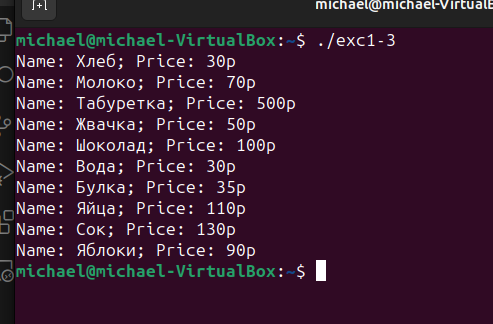


Рисунок 4. Текст к упражнению 3



Содержимое текстового файла test.txt (рис.6):

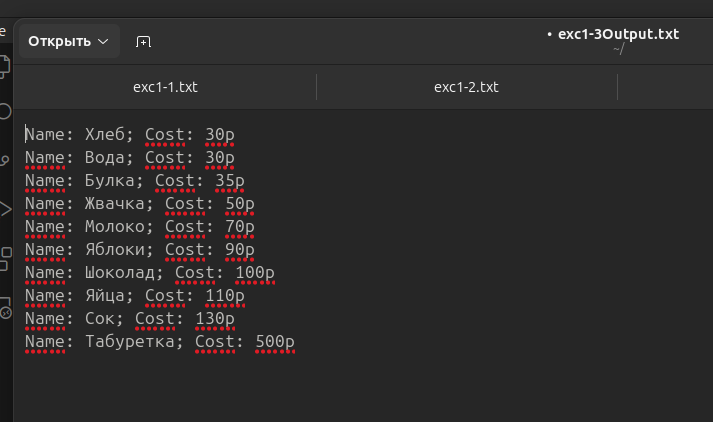


Рисунок 6. Текст после обработки программой

# Упражнение 4

## 4.1. Постановка задачи

Реализовать набор функций по работе со стеком. Для реализации стека воспользоваться структурой: следующий элемент стека хранит ссылку на предыдущий и значение.

· Реализовать добавление значения в стек.

· Реализовать извлечение значения из стека.

· Реализовать просмотр верхнего значения стека.

· Учесть при работе возможность переполнения стека.

## 4.2. Код программы

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

// Структура для представления узла стека

struct StackNode {

int data; // Данные узла стека (значение элемента)

struct StackNode\* next; // Указатель на следующий узел в стеке

};

// Структура для представления стека

struct Stack {

struct StackNode\* top; // Указатель на верхний элемент стека

int capacity; // Вместимость стека (максимальное количество элементов)

int size; // Текущий размер стека (количество элементов)

};

// Прототипы функций для работы со стеком

void sAddElem(int value, struct Stack\* stack);

int sPopElem(struct Stack\* stack);

int sPeekElem(struct Stack\* stack);

void initStack(struct Stack\* stack, int capacity);

struct StackNode\* sCreateElem(int data);

int main(void) {

// Простой тест для структуры стека

struct Stack stack;

initStack(&stack, 10);

printf("Adding elements to the stack:\n");

sAddElem(10, &stack);

printf("Top element: %d\n", sPeekElem(&stack));

sAddElem(13, &stack);

printf("Top element: %d\n", sPeekElem(&stack));

sAddElem(18, &stack);

printf("Top element: %d\n", sPeekElem(&stack));

printf("\nTrying to add more elements:\n");

sAddElem(19, &stack);

printf("Top element: %d\n", sPeekElem(&stack));

sAddElem(11, &stack);

printf("Top element: %d\n", sPeekElem(&stack));

printf("\nRemoving elements from the stack:\n");

printf("Popped element: %d\n", sPopElem(&stack));

printf("Top element: %d\n", sPeekElem(&stack));

printf("Adding a new element:\n");

sAddElem(22, &stack);

printf("Top element: %d\n", sPeekElem(&stack));

return 0;

}

// Функция для добавления элемента в стек

void sAddElem(int value, struct Stack \*stack) {

if (stack->size + 1 > stack->capacity) {

perror("Stack Overflow Exception"); // Вывод сообщения об ошибке переполнения стека

exit(EXIT\_FAILURE);

}

struct StackNode \*newNode = sCreateElem(value); // Создание нового узла стека

newNode->next = stack->top; // Установка указателя next нового узла на текущий верхний узел

stack->top = newNode; // Установка нового узла как верхний элемент стека

stack->size++; // Увеличение размера стека

}

// Функция для извлечения элемента из стека

int sPopElem(struct Stack \*stack) {

if (stack->top == NULL) {

perror("Stack is empty"); // Вывод сообщения об ошибке, если стек пуст

exit(EXIT\_FAILURE);

}

struct StackNode\* temp = stack->top; // Временная переменная для хранения верхнего узла

int data = temp->data; // Сохранение значения верхнего узла

stack->top = temp->next; // Установка указателя верхнего узла на следующий узел

free(temp); // Освобождение памяти, занятой верхним узлом

stack->size--; // Уменьшение размера стека

return data; // Возвращение значения верхнего узла

}

// Функция для просмотра верхнего элемента стека

int sPeekElem(struct Stack \*stack) {

if (stack->top == NULL) {

perror("Stack is empty"); // Вывод сообщения об ошибке, если стек пуст

exit(EXIT\_FAILURE);

}

return stack->top->data; // Возвращение значения верхнего узла без его извлечения

}

// Функция для инициализации стека

void initStack(struct Stack \*stack, int capacity) {

stack->top = NULL; // Установка верхнего элемента стека в NULL (стек пуст)

stack->capacity = capacity; // Установка вместимости стека

stack->size = 0; // Начальный размер стека установлен в 0

}

// Функция для создания нового узла стека

struct StackNode \*sCreateElem(int data) {

struct StackNode\* newNode = malloc(sizeof(struct StackNode)); // Выделение памяти под новый узел

if (newNode == NULL) {

perror("Failed to allocate memory"); // Вывод сообщения об ошибке, если не удалось выделить память

exit(EXIT\_FAILURE);

}

newNode->data = data; // Установка значения нового узла

newNode->next = NULL; // Установка указателя next в NULL (новый узел становится верхним)

return newNode; // Возвращение нового узла

}

## 4.3. Пример работы программы

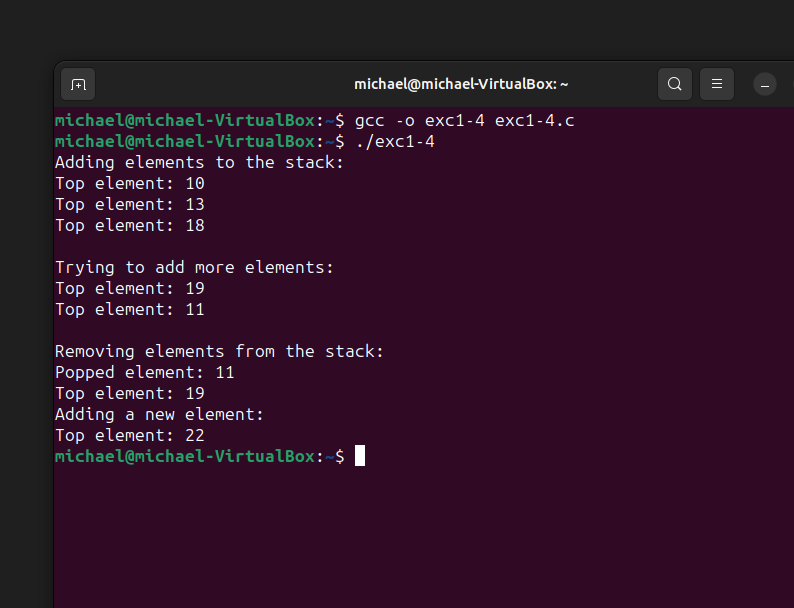


Рисунок 7. Пример работы программы упражнения 4

# Упражнение 5

## 5.1 Постановка задачи

Написать программу, вычисляющую значение выражения, записанного в постфиксной (обратной польской) записи, считываемого из входного файла. Считать, что выражение может содержать только цифры и знаки «+», «-», «\*» и «/». Примечание: для реализации воспользоваться функциями работы со стеком из предыдущего упражнения.

## 5.2 Код программы

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

#include <ctype.h>

// Объявление функции sCreateElem перед использованием

struct StackNode \*sCreateElem(int data);

// Структура для представления узла стека

struct StackNode {

int data; // Данные узла

struct StackNode\* next; // Ссылка на следующий узел

};

// Структура для представления стека

struct Stack {

struct StackNode\* top; // Вершина стека

int capacity; // Максимальная вместимость стека

int size; // Текущий размер стека

};

// Добавление элемента в стек

void sAddElem(int value, struct Stack \*stack) {

// Проверка на переполнение стека

if (stack->size + 1 > stack->capacity) {

perror("Stack Overflow Exception");

exit(EXIT\_FAILURE);

}

// Создание нового узла

struct StackNode \*newNode = sCreateElem(value);

// Установка ссылки на предыдущий верхний узел

newNode->next = stack->top;

// Установка нового узла как вершины стека

stack->top = newNode;

// Увеличение размера стека

stack->size++;

}

// Извлечение элемента из стека

int sPopElem(struct Stack \*stack) {

// Проверка на пустоту стека

if (stack->top == NULL) {

perror("Stack is empty");

exit(EXIT\_FAILURE);

}

// Сохранение верхнего узла

struct StackNode\* temp = stack->top;

// Получение данных из верхнего узла

int data = temp->data;

// Обновление вершины стека

stack->top = temp->next;

// Освобождение памяти верхнего узла

free(temp);

// Уменьшение размера стека

stack->size--;

// Возвращение извлеченных данных

return data;

}

// Получение значения верхнего элемента стека без его удаления

int sPeekElem(struct Stack \*stack) {

// Проверка на пустоту стека

if (stack->top == NULL) {

perror("Stack is empty");

exit(EXIT\_FAILURE);

}

// Возвращение данных верхнего узла

return stack->top->data;

}

// Инициализация стека

void initStack(struct Stack \*stack, int capacity) {

// Начальные значения стека

stack->top = NULL;

stack->capacity = capacity;

stack->size = 0;

}

// Создание узла стека

struct StackNode \*sCreateElem(int data) {

// Выделение памяти под новый узел

struct StackNode\* newNode = malloc(sizeof(struct StackNode));

// Проверка успешности выделения памяти

if (newNode == NULL) {

perror("Failed to allocate memory");

exit(EXIT\_FAILURE);

}

// Инициализация данных нового узла

newNode->data = data;

newNode->next = NULL;

// Возвращение созданного узла

return newNode;

}

// Проверка, является ли символ оператором

int isOperator(char c) {

return (c == '+' || c == '-' || c == '\*' || c == '/');

}

// Выполнение операции для двух операндов и оператора

int performOperation(char operator, int operand1, int operand2) {

switch (operator) {

case '+':

return operand1 + operand2;

case '-':

return operand1 - operand2;

case '\*':

return operand1 \* operand2;

case '/':

// Проверка деления на ноль

if (operand2 == 0) {

perror("Division by zero");

exit(EXIT\_FAILURE);

}

return operand1 / operand2;

default:

// Некорректный оператор

perror("Invalid operator");

exit(EXIT\_FAILURE);

}

}

// Оценка выражения в постфиксной записи

int evaluatePostfixExpression(FILE \*file) {

// Инициализация стека

struct Stack stack;

initStack(&stack, 100);

char token;

// Чтение символов из файла до конца

while (fscanf(file, " %c", &token) != EOF) {

if (isdigit(token)) {

// Если символ - число, помещаем его в стек

sAddElem(token - '0', &stack);

} else if (isOperator(token)) {

// Если символ - оператор, выполняем операцию

int operand2 = sPopElem(&stack);

int operand1 = sPopElem(&stack);

int result = performOperation(token, operand1, operand2);

// Результат помещаем обратно в стек

sAddElem(result, &stack);

} else {

// Некорректный символ в выражении

perror("Invalid character in input");

exit(EXIT\_FAILURE);

}

}

// Проверка корректности выражения

if (stack.size != 1) {

perror("Invalid expression");

exit(EXIT\_FAILURE);

}

// Возвращение результата выражения

return sPopElem(&stack);

}

// Главная функция

int main() {

// Открытие файла с постфиксным выражением

FILE \*file = fopen("exc4.txt", "r");

if (file == NULL) {

perror("Error opening file");

return EXIT\_FAILURE;

}

// Оценка постфиксного выражения и вывод результата

int result = evaluatePostfixExpression(file);

printf("Result: %d\n", result);

// Закрытие файла

fclose(file);

return EXIT\_SUCCESS;

}

## 5.3 Пример работы программы

Содержимое текстового файла test.txt (рис.8):

5 2 7 \* +

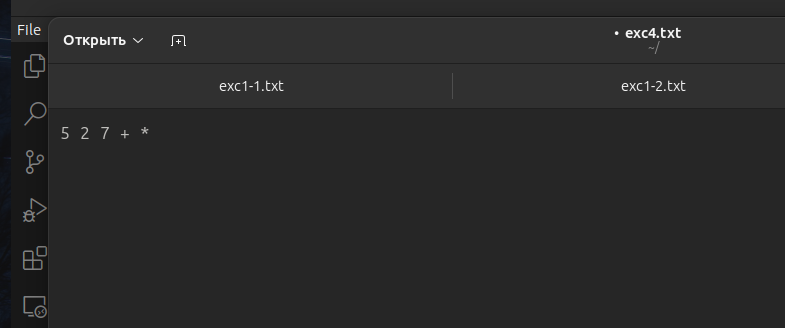


Рисунок 8. Текст к упражнению 5

На рис. 9 изображена работа программы для упражнения 5.

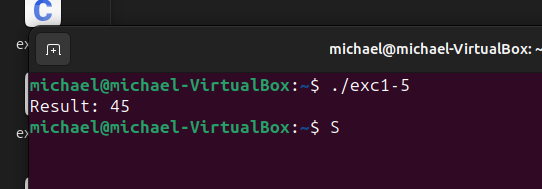


Рисунок 9. Пример работы программы упражнения 5

# Упражнение 6

## 6.1 Постановка задачи

Написать программу, реализующую перевод инфиксной записи арифметического выражения в постфиксную. Исходное арифметическое выражение состоит из цифр, знаков «+», «-», «\*», «/» и скобок. При реализации воспользоваться предиктивным анализатором и схемой трансляции из лекции о простом однопроходном компиляторе. Леворекурсивная грамматика для арифметических выражений представлена на слайде 9 лекции. Для реализации программы следует на ее основе

· составить схему трансляции

· заменить леворекурсивную грамматику равнозначной праворекурсивной

· реализовать предиктивный анализатор по составленной грамматике

## 6.2 Код программы

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

// Структура для представления узла стека

struct StackNode {

char data[10]; // Данные узла стека (значение элемента)

struct StackNode\* next; // Указатель на следующий узел в стеке

};

// Структура для представления стека

struct Stack {

struct StackNode\* top; // Указатель на верхний элемент стека

int capacity; // Вместимость стека (максимальное количество элементов)

int size; // Текущий размер стека (количество элементов)

};

// Прототипы функций для работы со стеком

void sAddElem(char\* value, struct Stack\* stack);

char\* sPopElem(struct Stack\* stack);

char\* sPeekElem(struct Stack\* stack);

void initStack(struct Stack\* stack, int capacity);

int getPriority(char\* op);

struct StackNode\* sCreateElem(char\* data);

// Функция для добавления элемента в стек

void sAddElem(char\* value, struct Stack \*stack) {

if (stack->size + 1 > stack->capacity) {

perror("Stack Overflow Exception");

exit(EXIT\_FAILURE);

}

struct StackNode \*newNode = sCreateElem(value);

newNode->next = stack->top;

stack->top = newNode;

stack->size++;

}

// Функция для извлечения элемента из стека

char\* sPopElem(struct Stack \*stack) {

if (stack->top == NULL) {

perror("Stack is empty");

exit(EXIT\_FAILURE);

}

struct StackNode\* temp = stack->top;

char\* data = strdup(temp->data); // Используем strdup для создания копии строки

stack->top = temp->next;

free(temp);

stack->size--;

return data;

}

// Функция для просмотра верхнего элемента стека

char\* sPeekElem(struct Stack \*stack) {

if (stack->top == NULL) {

perror("Stack is empty");

exit(EXIT\_FAILURE);

}

return stack->top->data;

}

// Функция для инициализации стека

void initStack(struct Stack \*stack, int capacity) {

stack->top = NULL;

stack->capacity = capacity;

stack->size = 0;

}

// Функция для создания нового узла стека

struct StackNode \*sCreateElem(char\* data) {

struct StackNode\* newNode = malloc(sizeof(struct StackNode));

if (newNode == NULL) {

perror("Failed to allocate memory");

exit(EXIT\_FAILURE);

}

strcpy(newNode->data, data);

newNode->next = NULL;

return newNode;

}

// Функция для перевода инфиксного выражения в постфиксное

void infixToPostfix(struct Stack\* operatorStack, struct Stack\* outputQueue, char\* infixExpression) {

char \*token = strtok(infixExpression, " ");

while (token != NULL) {

if (isdigit(token[0]) || isalpha(token[0])) {

// Если токен - операнд, добавляем его в выходную очередь

sAddElem(token, outputQueue);

} else if (strcmp(token, "(") == 0) {

// Если токен - открывающая скобка, помещаем его в стек операторов

sAddElem(token, operatorStack);

} else if (strcmp(token, ")") == 0) {

// Если токен - закрывающая скобка, извлекаем операторы из стека до открывающей скобки

while (strcmp(sPeekElem(operatorStack), "(") != 0) {

sAddElem(sPopElem(operatorStack), outputQueue);

}

// Извлекаем открывающую скобку из стека

sPopElem(operatorStack);

} else {

// Если токен - оператор, обрабатываем его в соответствии с приоритетом

while (operatorStack->top != NULL && strcmp(sPeekElem(operatorStack), "(") != 0 &&

getPriority(token) <= getPriority(sPeekElem(operatorStack))) {

sAddElem(sPopElem(operatorStack), outputQueue);

}

sAddElem(token, operatorStack);

}

token = strtok(NULL, " ");

}

// Извлекаем оставшиеся операторы из стека в выходную очередь

while (operatorStack->top != NULL) {

sAddElem(sPopElem(operatorStack), outputQueue);

}

}

// Вспомогательная функция для определения приоритета оператора

int getPriority(char\* op) {

if (strcmp(op, "\*") == 0 || strcmp(op, "/") == 0) {

return 2;

} else if (strcmp(op, "+") == 0 || strcmp(op, "-") == 0) {

return 1;

} else {

return 0; // Предполагаем, что все остальные операторы имеют более низкий приоритет

}

}

int main() {

struct Stack operatorStack, outputQueue;

initStack(&operatorStack, 100);

initStack(&outputQueue, 100);

char infixExpression[100]; // Предполагаем, что пользователь не введет более 100 символов

printf("Введите инфиксное выражение: ");

scanf("%99[^\n]", infixExpression); // Считываем строку до символа новой строки или 99 символов (не включая символ новой строки)

infixToPostfix(&operatorStack, &outputQueue, infixExpression);

// Выводим результат в постфиксной форме

printf("Postfix Expression: ");

while (outputQueue.top != NULL) {

printf("%s ", sPopElem(&outputQueue));

}

printf("\n");

return 0;

}

## 6.3 Пример работы программы

На рис. 12 изображена работа программы для упражнения 6.

Рисунок 12. Пример работы программы упражнения 4

Рисунок 12. Проверка результата с помощью https://www.semestr.online/