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 С»

по дисциплине «Языки программирования и методы трансляции»

Выполнила: студентка группы ИСТ-22-1б Афонина А. М

Проверил: Батин Сергей Евгеньевич

Пермь, 2023

**Содержание**

[Задание 1 3](#__RefHeading___Toc5704_3569584462)

[1.1. Постановка задачи 3](#__RefHeading___Toc5706_3569584462)

[1.2. Решение задачи, код программы 3](#__RefHeading___Toc5708_3569584462)

[1.3. Тестирование работы программы 4](#__RefHeading___Toc5710_3569584462)

[Задание 2 5](#__RefHeading___Toc5712_3569584462)

[2.1. Постановка задачи 5](#__RefHeading___Toc5714_3569584462)

[2.2. Решение задачи, код программы 5](#__RefHeading___Toc5716_3569584462)

[2.3. Тестирование работы программы 7](#__RefHeading___Toc5718_3569584462)

[Задание 3 7](#__RefHeading___Toc5720_3569584462)

[3.1. Постановка задачи 7](#__RefHeading___Toc1057_2420135925)

[3.2. Решение задачи, код программы 8](#__RefHeading___Toc5724_3569584462)

[3.3. Тестирование работы программы 9](#__RefHeading___Toc5726_3569584462)

[Задание 4 10](#__RefHeading___Toc5728_3569584462)

[4.1. Постановка задачи 10](#__RefHeading___Toc5730_3569584462)

[4.2. Решение задачи, код программы 10](#__RefHeading___Toc5732_3569584462)

[4.3. Тестирование работы программы 13](#__RefHeading___Toc5734_3569584462)

[Задание 5 13](#__RefHeading___Toc5736_3569584462)

[5.1. Постановка задачи 13](#__RefHeading___Toc5738_3569584462)

[5.2. Решение задачи, код программы 13](#__RefHeading___Toc5740_3569584462)

[5.3. Тестирование работы программы 16](#__RefHeading___Toc5742_3569584462)

[Задание 6 16](#__RefHeading___Toc1059_2420135925)

[6.1. Постановка задачи 16](#__RefHeading___Toc5738_35695844621)

[6.2. Решение задачи, схема трансляции 17](#__RefHeading___Toc5740_35695844621)

[6.3. Решение задачи, код программы 17](#__RefHeading___Toc5740_356958446212)

[6.4. Тестирование работы программы 19](#__RefHeading___Toc5742_35695844621)

# Задание 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. Решение задачи, код программы

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
4. **int** sizeofNum(**int** num){ //Счетчик цифр в числе
5. **int** numb = num;
6. **int** count = 0;
7. **while**(numb != 0){
8. numb /= 10;
9. count++;
10. }
11. **return** count;
12. }
13. **int** main() {
14. **int** i;
15. **int** len,num;
16. **int**\* mas = (**int**\*)malloc(len\***sizeof**(**int**));
17. printf("vvedite dlinu ");
18. scanf("%d", &len);
19. **for**(i = 0; i < len; i++) {
20. num = rand();
21. mas[i] = num;
22. printf("%d ", num);
23. }
25. FILE \*file = fopen("2\_1.txt", "w");
26. **if** (file == NULL) {
27. perror("Ошибка открытия файла!");
28. } **else** {
29. //Считывание данных с массива в обратном порядке
30. **for**(**int** i = len-1; i >= 0; i--){
31. //Счетчик цифр в числе
32. **int** count = sizeofNum(mas[i]);
33. //Перевод числа mas[i] в массив символов num\_char
34. **char** num\_char[count];
35. sprintf(num\_char, "%d", mas[i]);
36. //Посимвольная запись чисел в файл
37. **for** (**int** j = 0; j < count; j++){
38. fputc(num\_char[j],file);
39. }
40. fprintf(file, "\n");
41. }
42. }
43. fclose(file);
44. free(mas);
45. }

## 1.3. Тестирование работы программы

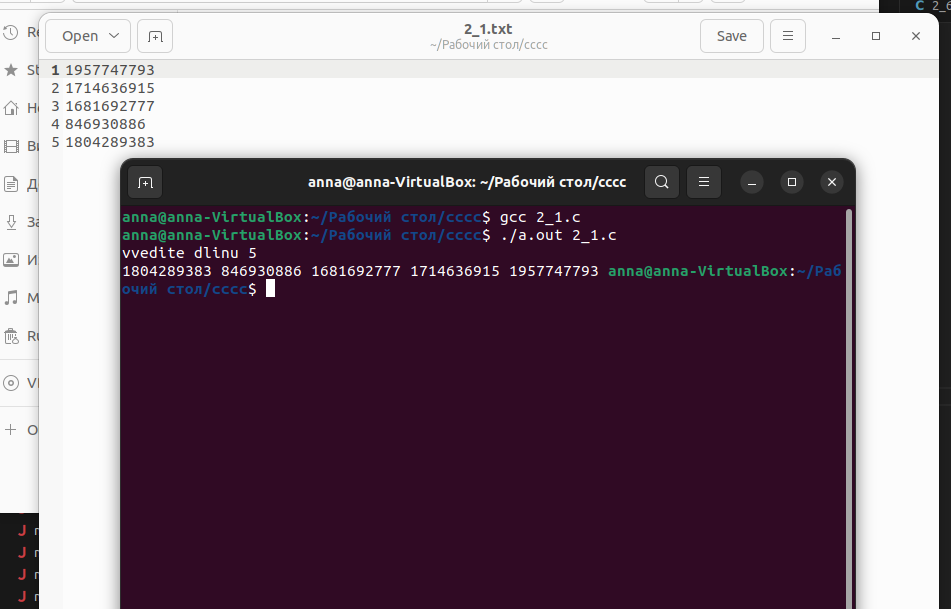


Рис.1.1 Тестирование программы ex1.c

# Задание 2

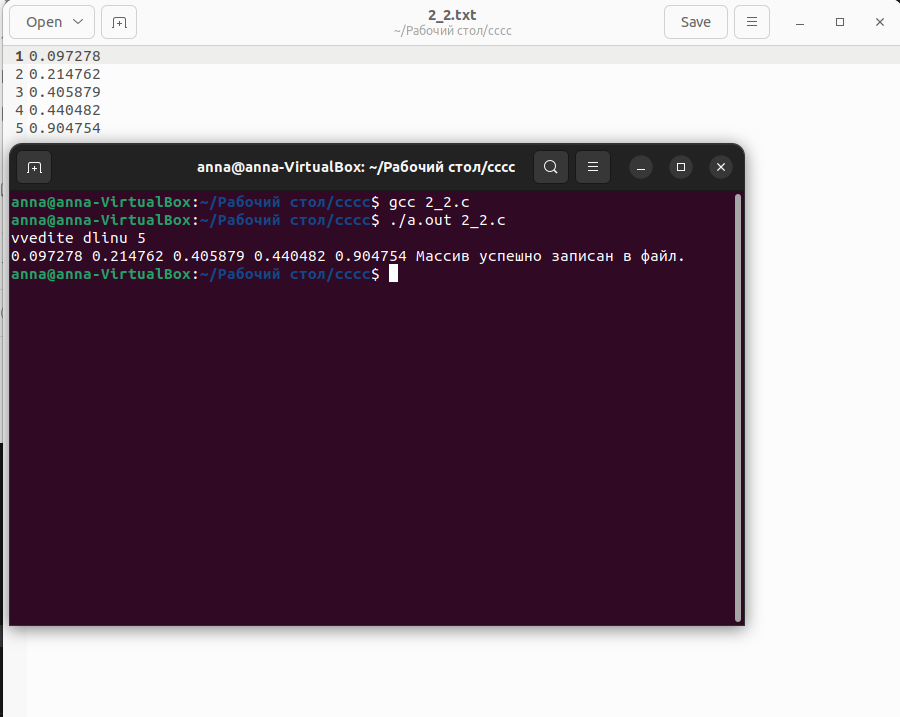
## 2.1. Постановка задачи

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. Решение задачи, код программы

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <time.h>
5. **int** main() {
6. srand(time(NULL));
7. **int** i;
8. **int** j;
9. **int** len;
10. **float** num;
11. **float** temp;
12. **float**\* mas = (**float**\*)malloc(len\***sizeof**(**float**));
13. printf("vvedite dlinu ");
14. scanf("%d", &len);
15. **for**(i = 0; i < len; i++) {
16. num = (**float**)rand() / RAND\_MAX;
17. mas[i] = num;
18. }
20. **for** (i = 0; i < len - 1; i++) {
21. **for** (j = 0; j < len - i - 1; j++) {
22. **if** (mas[j] > mas[j + 1]) {
23. temp = mas[j];
24. mas[j] = mas[j + 1];
25. mas[j + 1] = temp;
26. }
27. }
28. }
30. // Вывод отсортированного массива
31. **for** (i = 0; i < len; i++) {
32. printf("%f ", mas[i]);
33. }
35. FILE \*file = fopen("2\_2.txt", "w");
36. **if** (file == NULL) {
37. perror("Ошибка открытия файла!");
38. **return** 1;
39. } **else** {
40. **for** (i = 0; i < len; i++) {
41. fprintf(file, "%f\n", mas[i]);
42. }
43. fclose(file);
44. printf("Массив успешно записан в файл.\n");
45. }
46. free(mas);
47. **return** 0;
48. }

## 2.3. Тестирование работы программы

Рис.2.1 Тестирование программы ex2.c

# Задание 3

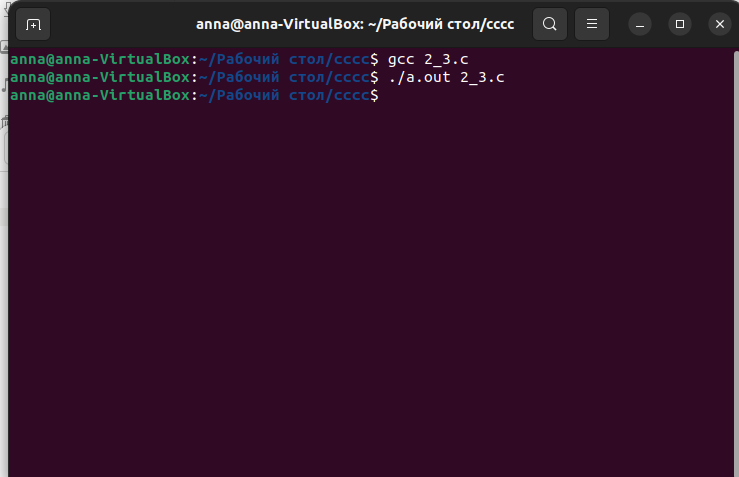
## 3.1. Постановка задачи

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. Решение задачи, код программы

1. #include <stdio.h>
2. #include <ctype.h>
4. **struct** products
5. {
6. **char** name[100];
7. **double** price;
8. };
10. **int** chislo\_strok(**char** \*filename){
11. FILE \*file = fopen(filename, "r");
12. **int** count = 0;
13. **if** (file == NULL) {
14. // Обработка ошибки при открытии файла
15. **return** -1;  // Возвращаем код ошибки
16. }**else**{
17. **char** sym;
18. **while** ((sym = fgetc(file)) != EOF) {
19. **if** (sym == '\n') {
20. count++;
21. }
22. }
23. }
25. fclose(file);
26. **return** count;
27. }
29. **void** sort ( **struct** products mas[]){
30. **int** len = chislo\_strok ("2\_3.txt");
31. **struct** products per;
32. **for** (**int** i = 0; i < len - 1; i++) {
33. **for** (**int** j = 0; j < len - i - 1; j++) {
34. **if** (mas[j].price > mas[j + 1].price) {
35. per = mas[j];
36. mas[j] = mas[j + 1];
37. mas[j + 1] = per;
38. }
39. }
40. }
41. }
43. **void** main(){
44. **int** chislo = chislo\_strok ("2\_3.txt");
45. **struct** products mas[chislo];
46. FILE \*file = fopen("2\_3.txt", "r");
47. **if** (file == NULL) {
48. perror("Ошибка открытия файла!");
49. } **else**{
50. **for** (**int** i = 0 ; i < chislo; i ++){
51. fscanf(file, "%100s %lf", mas[i].name, & mas[i].price);
53. }
54. fclose(file);
55. }
56. file = fopen("2\_31.txt","w");
57. sort(mas);
58. **if** (file == NULL) {
59. perror("Ошибка открытия файла!");
60. } **else** {
61. //запись отсортированных данных в файл
62. **for** (**int** i = 0; i < chislo; i++) {
63. fprintf(file, "%s %.2f\n", mas[i].name, mas[i].price);
64. }
65. }
66. fclose(file);
67. }

## 3.3. Тестирование работы программы

Рис.3.1 Тестирование программы ex3.c

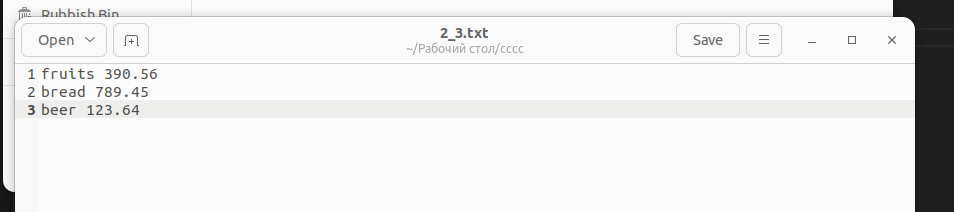


Рис.3.2 Файл с данными

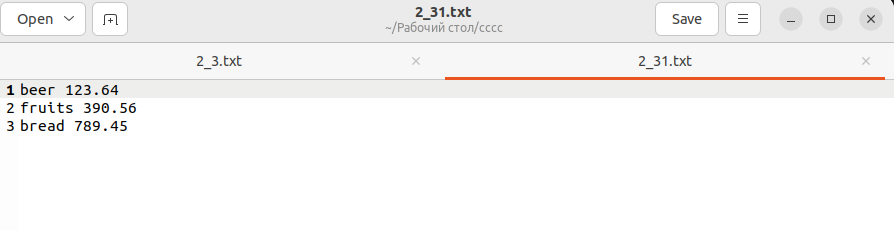


Рис.3.3 Файл результат работы программы (отсортированный)

# Задание 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. Решение задачи, код программы

1. #include <stdio.h>
2. #include <stdlib.h>
4. // Структура для элемента стека
5. typedef **struct** Node {
6. **int** value;
7. **struct** Node\* prev;
8. } Node;
10. // Глобальный указатель на вершину стека
11. Node\* top = NULL;
13. // Функция для добавления значения в стек
14. **void** push(**int** value) {
15. // Создаем новый узел для значения
16. Node\* newNode = (Node\*)malloc(**sizeof**(Node));
17. **if** (newNode == NULL) {
18. // Обработка ошибки при переполнении стека
19. printf("Ошибка: переполнение стека\n");
20. **return**;
21. }
23. // Заполняем данные нового узла
24. newNode->value = value;
25. newNode->prev = top;
27. // Обновляем указатель на вершину стека
28. top = newNode;
30. printf("Значение %d добавлено в стек\n", value);
31. }
33. // Функция для извлечения значения из стека
34. **int** pop() {
35. **if** (top == NULL) {
36. // Обрабатываем ошибку при попытке извлечения из пустого стека
37. printf("Ошибка: стек пуст\n");
38. **return** -1;
39. }
41. // Получаем значение верхнего элемента
42. **int** value = top->value;
44. // Обновляем указатель на вершину стека
45. Node\* prevTop = top;
46. top = top->prev;
48. // Освобождаем память, занятую предыдущим верхним элементом
49. free(prevTop);
51. **return** value;
52. }
54. // Функция для просмотра верхнего значения стека
55. **int** peek() {
56. **if** (top == NULL) {
57. // Обрабатываем ошибку при попытке просмотра пустого стека
58. printf("Ошибка: стек пуст\n");
59. **return** -1;
60. }
62. **return** top->value;
63. }
65. **int** main() {
66. // Пример использования функций для работы со стеком
68. // Добавляем значения в стек
69. push(5);
70. push(10);
71. push(15);
73. // Просматриваем верхнее значение стека
74. printf("Верхнее значение стека: %d\n", peek());
76. // Извлекаем значения из стека
77. **int** value1 = pop();
78. **int** value2 = pop();
79. **int** value3 = pop();
81. // Выводим извлеченные значения
82. printf("Извлеченные значения: %d, %d, %d\n", value1, value2, value3);
84. // Пытаемся извлечь значение из пустого стека
85. **int** value4 = pop();
87. **return** 0;
88. }

## 4.3. Тестирование работы программы

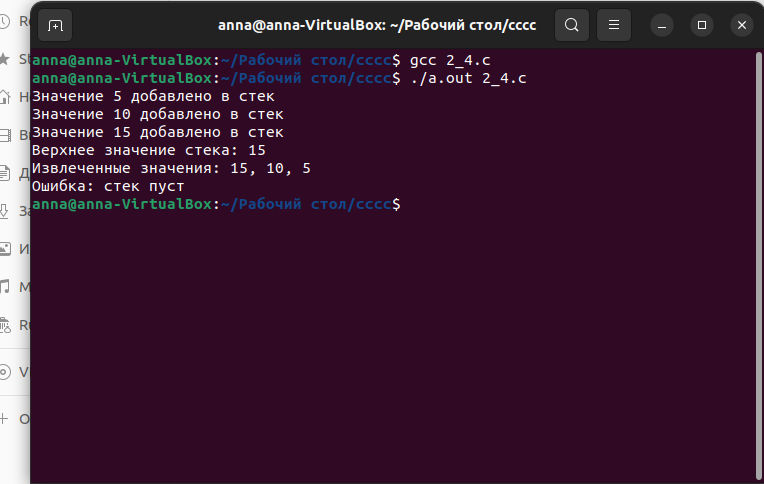


Рис. 4 Тестирование программы ex4.c

# Задание 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. Решение задачи, код программы

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <ctype.h>
5. // Структура для элемента стека
6. typedef **struct** Node {
7. **int** value;
8. **struct** Node\* prev;
9. } Node;
11. // Глобальный указатель на вершину стека
12. **struct** Stack {
13. **struct** Node\* top;
14. };
16. // Функция для создания пустого стека
17. **struct** Stack\* createStack() {
18. **struct** Stack\* stack = (**struct** Stack\*)malloc(**sizeof**(**struct** Stack));
19. stack->top = NULL;
20. **return** stack;
21. }
23. // Функция для добавления значения в стек
24. **void** push(**struct** Stack\* stack, **int** value) {
25. // Создаем новый узел для значения
26. Node\* newNode = (Node\*)malloc(**sizeof**(Node));
27. **if** (newNode == NULL) {
28. // Обработка ошибки при переполнении стека
29. printf("Ошибка: переполнение стека\n");
30. **return**;
31. }
33. // Заполняем данные нового узла
34. newNode->value = value;
35. newNode->prev = stack->top;
37. // Обновляем указатель на вершину стека
38. stack->top = newNode;
40. printf("Значение %d добавлено в стек\n", value);
41. }
43. // Функция для извлечения значения из стека
44. **int** pop(**struct** Stack\* stack) {
45. **if** (stack->top == NULL) {
46. // Обрабатываем ошибку при попытке извлечения из пустого стека
47. printf("Ошибка: стек пуст\n");
48. **return** -1;
49. }
51. // Получаем значение верхнего элемента
52. **int** value = stack->top->value;
54. // Обновляем указатель на вершину стека
55. Node\* prevTop = stack->top;
56. stack->top = stack->top->prev;
58. // Освобождаем память, занятую предыдущим верхним элементом
59. free(prevTop);
61. **return** value;
62. }
64. // Функция для просмотра верхнего значения стека
65. **int** peek(**struct** Stack\* stack) {
66. **if** (stack->top == NULL) {
67. // Обрабатываем ошибку при попытке просмотра пустого стека
68. printf("Ошибка: стек пуст\n");
69. **return** -1;
70. }
72. **return** stack->top->value;
73. }
75. **int** main() {
76. FILE \*file = fopen("2\_5.txt", "r");
78. **if** (file == NULL) {
79. printf("Ошибка при открытии файла\n");
80. **return** -1;
81. }
83. **struct** Stack\* stack = createStack();
85. **char** c;
86. **while** ((c = fgetc(file)) != EOF) {
87. **if** (isdigit(c)) {
88. push(stack, c - '0');
89. } **else** **if** (c == '+' || c == '-' || c == '\*' || c == '/') {
90. **int** operand2 = pop(stack);
91. **int** operand1 = pop(stack);
93. **switch** (c) {
94. **case** '+':
95. push(stack, operand1 + operand2);
96. **break**;
97. **case** '-':
98. push(stack, operand1 - operand2);
99. **break**;
100. **case** '\*':
101. push(stack, operand1 \* operand2);
102. **break**;
103. **case** '/':
104. push(stack, operand1 / operand2);
105. **break**;
106. }
107. }
108. }
110. fclose(file);
112. **if** (stack->top != NULL) {
113. **int** result = pop(stack);
114. printf("Результат: %d\n", result);
115. } **else** {
116. printf("Ошибка: стек пуст\n");
117. }
119. free(stack);
121. **return** 0;
122. }

## 5.3. Тестирование работы программы

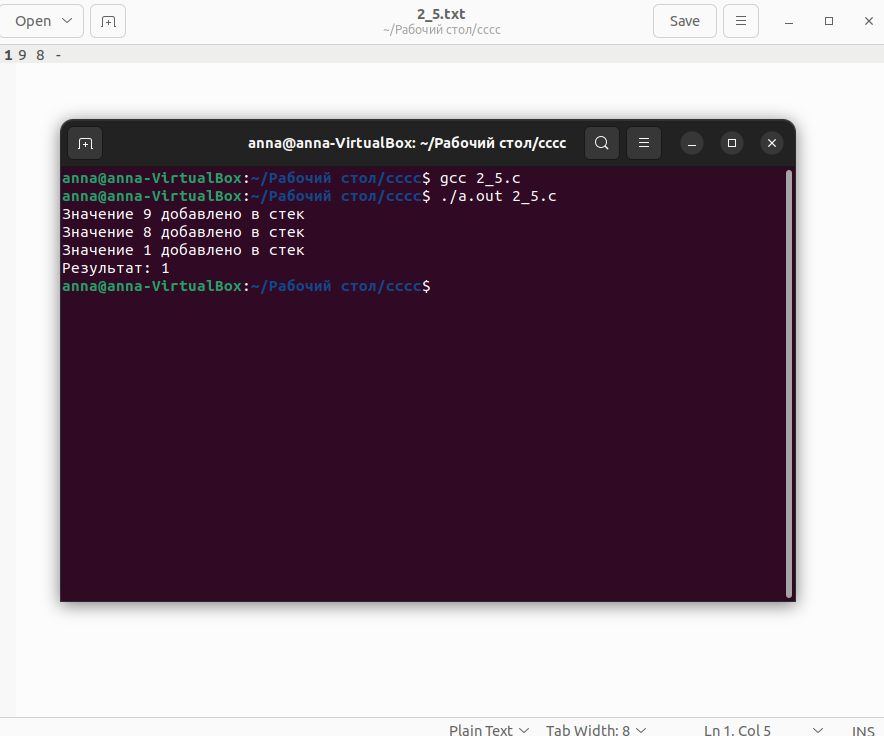


Рис. 5.2 Тестирование работы программы ex5.c

# Задание 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. Решение задачи, схема трансляции

*factor → (expr)*

*|* *digit* {*post[index] := digit || index++* }

*term →* *factor more**Term*

*moreTerm→ \* factor* { *post[index] := ‘\*’ || index++* } *moreTerm*

*| / factor* { *post[index] := ‘/’* *|| index++* } *moreTerm*

*| ε*

*expr → term more**Expr*

*moreExpr→ + term* { *post[index] := ‘+’ || index++*} *moreExpr*

*| - term* { *post[index] := ‘-’ || index++* } *moreExpr*

*| ε*

*parse → expr* { *print(expr)* }

## 6.3. Решение задачи, код программы

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#define n 100

char infExpr[n]; //инфиксное выражение

int infPos = 0; //номер символа инфиксного выражения

char postExpr[n];//постфиксное выражение

int postPos = 0; //номер символа постфиксного выражения

void factor() { //нетерминал продукции скобок и цифр

if (isdigit(infExpr[infPos])) {

postExpr[postPos++] = infExpr[infPos++];

} else if (infExpr[infPos] == '(') {

infPos++;

expr();

if (infExpr[infPos] == ')') {

infPos++;

} else {

perror("Ошибка расположения скобок");

exit(0);

}

} else {

perror("Встреча неожиданного символа");

exit(0);

}

}

void moreTerm() { //нетерминал продукций с операциями '\*' и '/'

if (infExpr[infPos] == '\*' || infExpr[infPos] == '/') {

char op = infExpr[infPos++];

factor();

moreTerm();

postExpr[postPos++] = op;

}

}

void term() { //нетерминал наивысшего приоритета продукций

factor(); // скобки - наивысший приоритет

moreTerm();// '\*' и '/' - повышенный приоритет

}

void moreExpr() { //нетерминал продукций с операциями '+' и '-'

if (infExpr[infPos] == '+' || infExpr[infPos] == '-') {

char op = infExpr[infPos++];

term();

moreExpr();

postExpr[postPos++] = op;

}

}

void expr() { //нетерминал формирования выражаения

term(); // наивысший приоритет

moreExpr();// нижайший приоритет

}

void parse() { //нетерминал перевода в постфиксную запись и вывода

expr();

printf("Постфиксное выражение: %s\n", postExpr);

}

int main() {

printf("Введите инфиксное выражение: ");

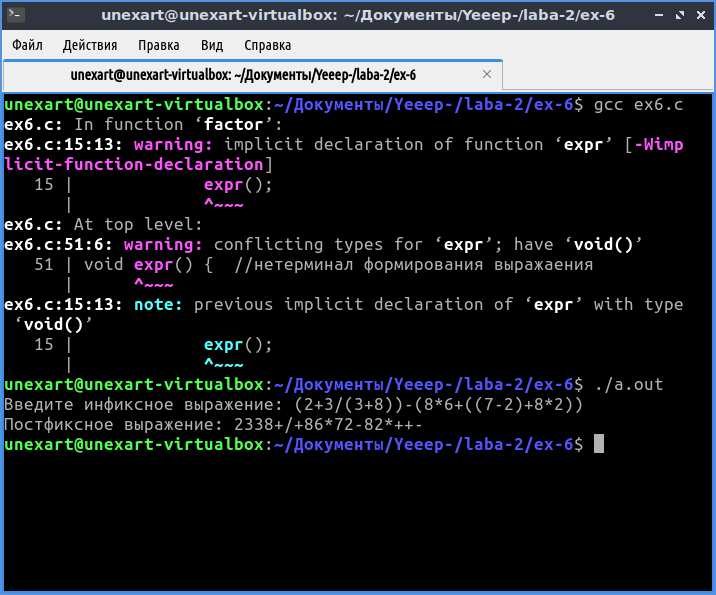
scanf("%s", infExpr);

parse();

return 0;

}

## 6.4. Тестирование работы программы

Рис. 6.1 Тестирование работы программы ex6.c