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>

int main(int argc, char \*argv[])

{

if (argc == 2) // проверка количества аргументов

{

FILE \*fp; // создание поинтера на файл

fp = fopen(argv[1], "r"); // открытие файла

if (fp == NULL) // проверка файла на существование

{

printf("Enter a file name: ./a.out filename.txt\n"); // вывод сообщения об ошибке

}

srand(time(NULL)); // инициализация генератора случайных чисел

int l = rand() % 20 + 20; // генерация размера массива

int ar[l]; // создание массива

for (int i = 0; i < l; i++) // цикл для заполнения массива случайными числами

{

ar[i] = rand(); // генерация случайного числа и запись в массив

printf("%i\n", ar[i]); // вывод мыссива

}

fp = fopen(argv[1], "w"); // открытие файла для записи

for (int i = 0; i < l; i++) // цикл для записи чисел из массива в файл

{

fprintf(fp, "%i", ar[l - i - 1]); // запись в файл случайного числа

fprintf(fp, "%s", "\n"); // запись в файл переноса строки

}

fclose(fp); // закрытие файла

}

else

{

puts("Enter a file name: ./a.out filename.txt\n"); // вывод сообщения об ошибке

}

return 0;

}

## 1.3 Пример работы программы

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

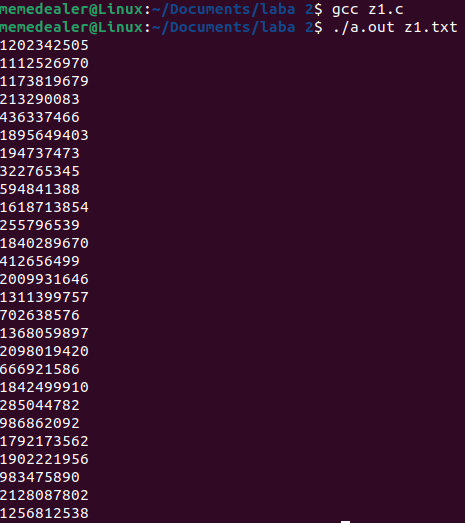


Рисунок 1. Результат программы 1 упражнения

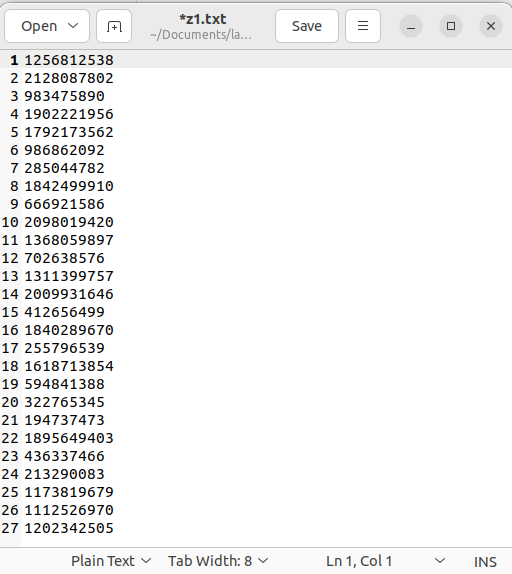


Рисунок 2. Результат программы 1 упражнения

# 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>

int main(int argc, char \*argv[])

{

if (argc == 2) // проверка количества аргументов

{

FILE \*fp; // создание поинтера на файл

fp = fopen(argv[1], "r"); // открытие файла

if (fp == NULL) // проверка файла на существование

{

printf("Enter a file name: ./a.out filename.txt\n"); // вывод сообщения об ошибке

}

srand(time(NULL)); // инициализация генератора случайных чисел

int l = rand() % 20 + 20; // генерация размера массива

double ar[l]; // создание массива

double r = 0; // создание переменной для генерации случайных чисел

for (int i = 0; i < l; i++) // цикл для заполнения массива

{

r = (double)(rand()) / (double)(rand()); // генерация случайного double

r = r - (int)r; // удаление целочисленной части из double

ar[i] = r; // запись числа в массив

printf("%lf\n", ar[i]); // вывод мыссива

}

// сортировка массива с помощью selection sort

double temp = 0; // переменная для сортировки

for (int i = 0; i < l; i++) // перебор всех чисел массива

{

for (int j = i + 1; j < l; j++) // перебор всех чисел массива, начиная с позиции i

{

if (ar[i] > ar[j]) // сравнение чисел

{

temp = ar[i]; // замена чисел при удовлетворении условию

ar[i] = ar[j];

ar[j] = temp;

}

}

}

fp = fopen(argv[1], "w"); // открытие файла для записи

for (int i = 0; i < l; i++) // цикл для записи чисел из массива в файл

{

fprintf(fp, "%lf", ar[i]); // запись в файл случайного числа

fprintf(fp, "%s", "\n"); // запись в файл переноса строки

}

fclose(fp); // закрытие файла

}

else

{

puts("Enter a file name: ./a.out filename.txt\n"); // вывод сообщения об ошибке

}

return 0;

}

## 2.3. Примеры работы программы

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

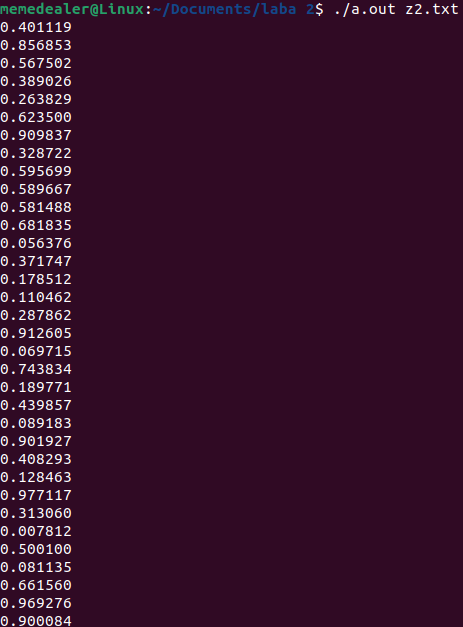


Рисунок 3. Пример работы программы упражнения 2

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

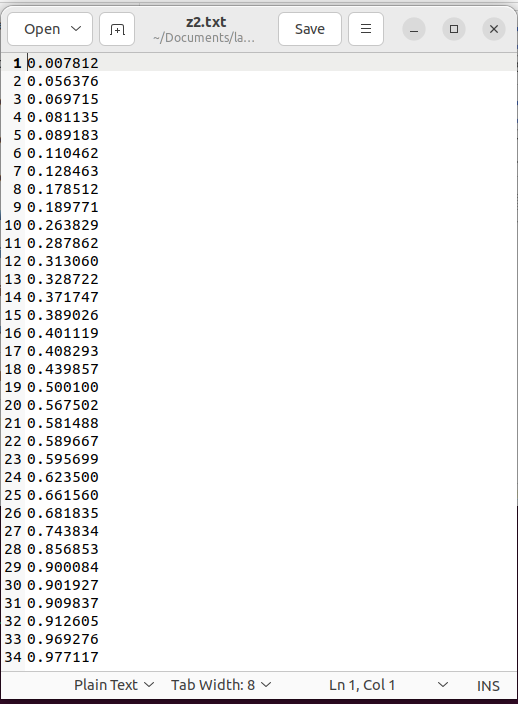


Рисунок 4. Результат программы 1 упражнения 2

# 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 <string.h>

#include <stdlib.h>

struct arr // структура данных для представления массива

{

int size; // емкость массива

int maxlength; // максимальная длина строки

char \*\*prodnames; // массив элементов названий

int \*prices; // массив элементов цен

};

struct arr \*newArray(int strc, int strl) // функция для инициализации массива

{

struct arr \*pt = (struct arr \*)malloc(sizeof(struct arr)); // выделение динамической памяти для массива

pt->size = strc; // переменная длины массива

pt->maxlength = strl; // переменная максимальной длины строки

pt->prodnames = (char \*\*)malloc(strc \* sizeof(char \*)); // выделение динамической памяти для массива элементов

pt->prices = (int \*)malloc(sizeof(int) \* strc); // выделение динамической памяти для массива элементов

return pt;

}

void selectionsort(struct arr \*pt) // сортировка массива с помощью selection sort

{

int tempprice = 0; // переменная для сортировки

char tempname[pt->maxlength]; // переменная для сортировки

for (int i = 0; i < pt->size; i++) // перебор всех чисел массива

{

for (int j = i + 1; j < pt->size; j++) // перебор всех чисел массива, начиная с позиции i

{

if (pt->prices[i] > pt->prices[j]) // сравнение чисел

{

tempprice = pt->prices[i]; // замена чисел при удовлетворении условию

pt->prices[i] = pt->prices[j];

pt->prices[j] = tempprice;

strcpy(tempname, pt->prodnames[i]); // замена строк при удовлетворении условию

strcpy(pt->prodnames[i], pt->prodnames[j]);

strcpy(pt->prodnames[j], tempname);

}

}

}

}

int \*checkstr(FILE \*fp, char \*filename) // функция для подсчета количества строк и максимальной длины строки

{

int strc = 1; // количество строк файла

int strl = 1; // максимальная длина строки файла

int curstrl = 1; // текущая длина строки файла

fp = fopen(filename, "r"); // открытие файла

while (!feof(fp)) // цикл для подсчета количества строк и макисмальной длины строки

{

if (fgetc(fp) == '\n') // чтение файла посимвольно и проверка на равенство переносу строки

{

strc++; // увеличение переменной

curstrl = 0; // обновление длины строки

}

else

{

curstrl++; // увеличение переменной

if (curstrl > strl) // сравнение текущей длины строки и максимальной

{

strl = curstrl; // обновление максимального длины строки

}

}

}

fclose(fp); // закрытие файла

int \*strings = malloc(2); // массив для вывода

strings[0] = strc; // добавление количества строк в массив

strings[1] = strl; // добавление длины строки в массив

return strings;

}

void toarray(struct arr \*pt, FILE \*fp, char \*filename) // функция для переноса файла в массив

{

char prodname[pt->size]; // строка для чтения

int price; // переменная для чтения

int i = 0; // переменная для перебора

fp = fopen(filename, "r"); // открытие файла

while (fscanf(fp, "%s %i", prodname, &price) > 0) // цикл для чтения файла

{

pt->prodnames[i] = prodname; // добавление элемента

pt->prices[i] = price; // добавление элемента

i++; // увеличение переменной

}

fclose(fp); // закрытие файла

}

void tofile(struct arr \*pt, FILE \*fp, char \*filename) // функция для переноса массива в файл

{

fp = fopen(filename, "w"); // открытие файла для записи

for (int i = 0; i < pt->size; i++) // перебор всех чисел массива

{

fprintf(fp, "%s %i\n", pt->prodnames[i], pt->prices[i]); // запись в файл

}

fclose(fp); // закрытие файла

}

int main(int argc, char \*argv[])

{

if (argc == 3) // проверка количества аргументов

{

FILE \*fp; // создание поинтера на файл

struct arr \*pt = newArray(checkstr(fp, argv[1])[0], checkstr(fp, argv[1])[1]); // создание массива емкостью strc

toarray(pt, fp, argv[1]); // добавление элементов из файла в массива

selectionsort(pt); // сортировка элементов массива

tofile(pt, fp, argv[2]); // вывод элементов из массива в файл

}

else

{

puts("Enter a file name: ./a.out filename.txt ./a.out filename.txt\n"); // вывод сообщения об ошибке

}

return 0;

}

## 3.3. Пример работы программы

Содержимое текстового файла test.txt (рис.5):

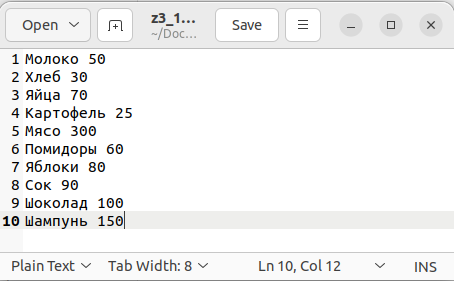


Рисунок 5. Текст к упражнению 3

На рис.6 представлен результат работы программы упражнения 3.

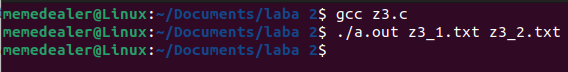


Рисунок 6. Пример работы программы упражнения 4

Содержимое текстового файла test.txt (рис.7):

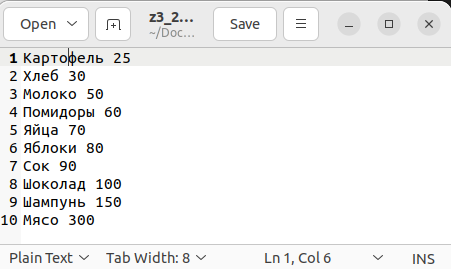


Рисунок 7. Текст после обработки программой

# 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>

struct stack // структура данных для представления stack

{

int maxsize; // максимальная емкость stack

int top; // индекс вершины stack

int \*items; // массив элементов stack

};

struct stack \*newStack(int capacity) // функция для инициализации stack

{

struct stack \*pt = (struct stack \*)malloc(sizeof(struct stack)); // выделение динамической памяти для stack

pt->maxsize = capacity; // определение емкости

pt->top = -1; // определение индекса вершины

pt->items = (int \*)malloc(sizeof(int) \* capacity); // выделение динамической памяти для массива элементов

return pt;

}

int size(struct stack \*pt) // функция для возврата размера stack

{

return pt->top + 1; // возвращение текущего индекса вершины

}

int isEmpty(struct stack \*pt) // функция для проверки, пуст stack или нет

{

return pt->top == -1; // возвращение равенства индекса вершины -1

}

int isFull(struct stack \*pt) // функция для проверки, заполнен ли stack или нет

{

return pt->top == pt->maxsize - 1; // возвращение максимальной емкости stack

}

void push(struct stack \*pt, int x) // функция для добавления элемента `x` в stack

{

if (isFull(pt)) // проверка, не заполнен ли уже stack

{

printf("Overflow\nProgram Terminated\n"); // вывод информации о том, что stack заполнен

exit(EXIT\_FAILURE); // завершение программы

}

printf("Inserting %d\n", x); // вывод информации о том, что элемент добавляется в stack

pt->items[++pt->top] = x; // добавление элемента и увеличиние индекса вершины

}

int peek(struct stack \*pt) // функция для возврата верхнего элемента stack

{

if (!isEmpty(pt)) // проверка на пустой stack

{

return pt->items[pt->top]; // возврат верхнего элемента

}

else

{

exit(EXIT\_FAILURE); // завершение программы

}

}

int pop(struct stack \*pt) // функция для извлечения верхнего элемента из stack

{

if (isEmpty(pt)) // проверка на пустой stack

{

printf("Underflow\nProgram Terminated\n"); // вывод информации о том, что stack пустой

exit(EXIT\_FAILURE); // завершение программы

}

printf("Removing %d\n", peek(pt)); // вывод сообщения о извлечении элемента

return pt->items[pt->top--]; // уменьшение размер stack на 1

}

int main()

{

struct stack \*pt = newStack(5); // создание stack емкостью 5

push(pt, 1); // добавление 1 в stack

push(pt, 2); // добавление 2 в stack

push(pt, 3); // добавление 3 в stack

printf("The top element is %d\n", peek(pt)); // вывод верхнего элемента

pop(pt); // удаление верхнего элемента из stack

pop(pt); // удаление верхнего элемента из stack

pop(pt); // удаление верхнего элемента из stack

if (isEmpty(pt)) // проверка, пуст stack или нет

{

printf("The stack is empty"); // вывод сообщения

}

else

{

printf("The stack is not empty"); // вывод сообщения

}

return 0;

}

## 4.3. Пример работы программы

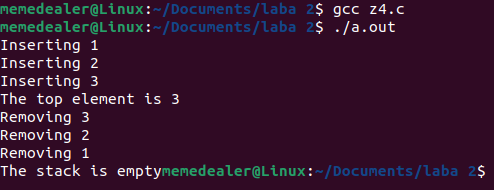


Рисунок 7. Пример работы программы упражнения 4

# 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 <ctype.h>

#include <string.h>

struct stack // структура данных для представления stack

{

int maxsize; // максимальная емкость stack

int top; // индекс вершины stack

double \*items; // массив элементов stack

};

struct stack \*newStack(int capacity) // функция для инициализации stack

{

struct stack \*pt = (struct stack \*)malloc(sizeof(struct stack)); // выделение динамической памяти для stack

pt->maxsize = capacity; // определение емкости

pt->top = -1; // определение индекса вершины

pt->items = (double \*)malloc(sizeof(double) \* capacity); // выделение динамической памяти для массива элементов

return pt;

}

void push(struct stack \*pt, double x) // функция для добавления элемента `x` в stack

{

if (pt->top == pt->maxsize - 1) // проверка, не заполнен ли уже stack

{

printf("Overflow\nProgram Terminated\n"); // вывод информации о том, что stack заполнен

exit(EXIT\_FAILURE); // завершение программы

}

pt->items[++pt->top] = x; // добавление элемента и увеличиние индекса вершины

}

double peek(struct stack \*pt) // функция для возврата верхнего элемента stack

{

if (pt->top == -1) // проверка на пустой stack

{

exit(EXIT\_FAILURE); // завершение программы

}

else

{

return pt->items[pt->top]; // возврат верхнего элемента

}

}

double pop(struct stack \*pt) // функция для извлечения верхнего элемента из stack

{

if (pt->top == -1) // проверка на пустой stack

{

printf("Underflow\nProgram Terminated\n"); // вывод информации о том, что stack пустой

exit(EXIT\_FAILURE); // завершение программы

}

return pt->items[pt->top--]; // уменьшение размер stack на 1

}

void polish(struct stack \*pt, FILE \*fp, char \*filename) // решение выражения в обратной польской записи

{

char ch; // переменная для чтения файла

char str[30] = ""; // строка для записи числа

char \*endptr; // символ, следующий за числом после выполнения функции strtod

fp = fopen(filename, "r"); // открытие файла

while (!feof(fp)) // цикл для чтения файла

{

ch = fgetc(fp); // посимвольное чтение файла

if (ch == ',') // проверка ch на равенство запятой

{

ch = '.'; // замена ch на точку

}

if (isdigit(ch) || ch == '.') // проверка символа на число

{

strncat(str, &ch, 1); // добавление char в строку

}

else

{

if (ch == ' ') // проверка символа на пробел

{

if (strlen(str) > 0) // проверка длины строки

{

double numb = strtod(str, &endptr); // генерация числа из строки

push(pt, numb); // добавление в стек

strcpy(str, ""); // обнуление строки

}

}

else if (ch == '+') // проверка символа на оператор +

{

double first = pop(pt); // извлечение элемента из стека

double second = pop(pt); // извлечение элемента из стека

push(pt, second + first); // занесение элемента в стек

}

else if (ch == '-') // проверка символа на оператор -

{

double first = pop(pt); // извлечение элемента из стека

double second = pop(pt); // извлечение элемента из стека

push(pt, second - first); // занесение элемента в стек

}

else if (ch == '\*') // проверка символа на оператор \*

{

double first = pop(pt); // извлечение элемента из стека

double second = pop(pt); // извлечение элемента из стека

push(pt, second \* first); // занесение элемента в стек

}

else if (ch == '/') // проверка символа на оператор /

{

double first = pop(pt); // извлечение элемента из стека

double second = pop(pt); // извлечение элемента из стека

push(pt, second / first); // занесение элемента в стек

}

}

}

fclose(fp); // закрытие файла

}

int numbinarow(FILE \*fp, char \*filename) // проверка максимального количество чисел в записи под ряд

{

fp = fopen(filename, "r"); // открытие файла

int curr = 2; // переменная для текущего количества чисел под ряд

int maximum = 2; // переменная для максимального количества чисел под ряд

while (!feof(fp)) // цикл для подсчета количества чисел под ряд

{

char ch = fgetc(fp); // чтение файла посимвольно

if (ch == ' ') // проверка на равенство пробелу

{

curr += 2; // увеличение переменной

if (curr > maximum) // сравнение текущей числа чисел под ряд и максимально

{

maximum = curr; // обновление максимального количества

}

}

else if (!isdigit(ch)) // проверка на неравенство символу

{

curr = 2; // обновление переменной

}

}

return maximum;

}

int main(int argc, char \*argv[])

{

if (argc == 2) // проверка количества аргументов

{

FILE \*fp; // создание поинтера на файл

struct stack \*pt = newStack(numbinarow(fp, argv[1])); // создание stack с вычислением емкости

push(pt, 0); // добавление в стек 0

polish(pt, fp, argv[1]); // вычисление выражения

printf("Result: %lf\n", peek(pt)); // вывод результата

}

return 0;

}

## 5.3 Пример работы программы

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

5 2 7 \* +

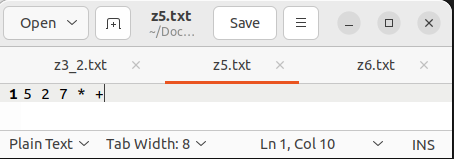


Рисунок 8. Текст к упражнению 4

На рис. 9 изображена работа программы для упражнения 4.



Рисунок 9. Пример работы программы упражнения 4



Рисунок 10. Проверка результата с помощью Google

Содержимое текстового файла test.txt (рис.11):

5 99.23 \* 93 4 92 43 9 - \* 2 / 15 \* - 2 - \* 5 2.65 - 34.54 3 7 4 \* + / \* / -

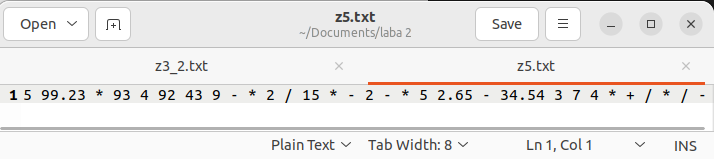


Рисунок 11. Текст к упражнению 6

На рис. 11 изображена работа программы для упражнения 6.



Рисунок 11. Пример работы программы упражнения 4



Рисунок 12. Проверка результата с помощью Google

# 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>

struct stack // структура данных для представления stack

{

int maxsize; // максимальная емкость stack

int top; // индекс вершины stack

char \*\*items; // массив элементов stack

};

struct stack \*newStack(int capacity) // функция для инициализации stack

{

struct stack \*pts = (struct stack \*)malloc(sizeof(struct stack)); // выделение динамической памяти для stack

pts->maxsize = capacity; // определение емкости

pts->top = -1; // определение индекса вершины

pts->items = (char \*\*)malloc(sizeof(char \*) \* capacity); // выделение динамической памяти для массива элементов

for (int i = 0; i < capacity; i++) // выделение динамической памяти для строк

{

pts->items[i] = (char \*)malloc(sizeof(char) \* 10);

}

return pts;

}

void stack\_push(struct stack \*pts, char \*x) // функция для добавления элемента `x` в stack

{

if (pts->top == pts->maxsize - 1) // проверка, не заполнен ли уже stack

{

printf("Overflow\nProgram Terminated\n"); // вывод информации о том, что stack заполнен

exit(EXIT\_FAILURE); // завершение программы

}

strcpy(pts->items[++pts->top], x); // добавление элемента и увеличиние индекса вершины

}

char \*stack\_peek(struct stack \*pts) // функция для возврата верхнего элемента stack

{

if (pts->top != -1) // проверка на пустой stack

{

return pts->items[pts->top]; // возврат верхнего элемента

}

else

{

exit(EXIT\_FAILURE); // завершение программы

}

}

char \*stack\_pop(struct stack \*pts) // функция для извлечения верхнего элемента из stack

{

if (pts->top == -1) // проверка на пустой stack

{

printf("Underflow\nProgram Terminated\n"); // вывод информации о том, что stack пустой

exit(EXIT\_FAILURE); // завершение программы

}

return pts->items[pts->top--]; // уменьшение размер stack на 1

}

struct queue

{

char \*\*items; // указатель на данные

int bottom; // указатель на нижнюю границу

int top; // указатель на верхнюю границу

int amount; // количество элементов в очереди

int maxsize; // максимальное количество элементов

};

struct queue \*newQueue(int capacity) // функция для инициализации очереди

{

struct queue \*ptq = (struct queue \*)malloc(sizeof(struct queue)); // выделение динамической памяти для очереди

ptq->maxsize = capacity; // определение емкости

ptq->bottom = -1; // определение нижней границы

ptq->top = -1; // определение верхней границы

ptq->amount = 0; // определение количества элементов

ptq->items = (char \*\*)malloc(sizeof(char \*) \* capacity); // выделение динамической памяти для массива элементов

for (int i = 0; i < capacity; i++) // выделение динамической памяти для строк

{

ptq->items[i] = (char \*)malloc(sizeof(char) \* 10);

}

return ptq;

}

void queue\_push(struct queue \*ptq, char \*x) // функция для добавления элемента `x` в очередь

{

if (ptq->amount == ptq->maxsize) // проверка не заполнена ли уже очередь

{

printf("Overflow\nProgram Terminated\n"); // вывод информации о том, что stack заполнен

exit(EXIT\_FAILURE); // завершение программы

}

strcpy(ptq->items[++ptq->top], x); // добавление элемента и увеличиние верхней границы

ptq->amount++; // увеличение количества элементов

if (ptq->top > ptq->maxsize - 1) // обнуление верхней границы при переходе за максимальный размер

{

ptq->top = -1;

}

}

char \*queue\_pop(struct queue \*ptq) // функция для извлечения верхнего элемента из очередь

{

if (ptq->amount == 0) // проверка на пустую очередь

{

printf("Underrflow\nProgram Terminated\n"); // вывод информации о том, что stack заполнен

exit(EXIT\_FAILURE); // завершение программы

}

char \*a = ptq->items[++ptq->bottom]; // присвоение элемента нижней границы

ptq->amount--; // уменьшение количества элементов

if (ptq->bottom > ptq->maxsize - 1) // обнуление нижней границы при переходе за максимальный размер

{

ptq->bottom = -1;

}

return a;

}

void polish(struct stack \*pts, struct queue \*ptq, char \*sum) // перевод выражения в обратную польскую запись

{

char ch; // переменная для чтения файла

char str[30] = ""; // строка для записи числа

char \*endptr; // символ, следующий за числом после выполнения функции strtod

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

{

ch = sum[i]; // посимвольное чтение файла

if (ch != ' ') // проверка символа на пробел

{

strncat(str, &ch, 1); // добавление char в строку

}

else

{

double numb = strtod(str, &endptr); // генерация числа из строки для проверки, чем она является

if (endptr != str) // если endptr не указывает на тот же адрес, что и str, то строка - число

{

queue\_push(ptq, str); // добавление числа в очередь

}

else

{

if (strcmp(str, "+") == 0 || strcmp(str, "-") == 0 || strcmp(str, "\*") == 0 || strcmp(str, "/") == 0) // проверка на равенство строки оператору "+-\*/"

{

if (pts->top == -1 || strcmp(str, "(") == 0) // проверка вершины стека на равенство "(" или на пустой стек

{

stack\_push(pts, str); // добавление оператора в стек

}

else if ((strcmp(str, "\*") == 0 || strcmp(str, "/") == 0) & (strcmp(stack\_peek(pts), "+") == 0 || strcmp(stack\_peek(pts), "-") == 0)) // сравнение приоритета оператора из вершины стека с оператором из строки

{

stack\_push(pts, str); // добавление оператора в стек

}

else // остальные случаи

{

while (!(pts->top == -1)) // цикл пока вершина стека не равна 0

{

if (strcmp(stack\_peek(pts), "(") == 0 || ((strcmp(str, "\*") == 0 || strcmp(str, "/") == 0) & ((strcmp(stack\_peek(pts), "+") == 0 || strcmp(stack\_peek(pts), "-") == 0)))) // проверка вершины на равенство "(" или оператору с проядком ниже

{

break; // конец цикла

}

queue\_push(ptq, stack\_pop(pts)); // добавление оператора в очередь

}

stack\_push(pts, str); // добавление оператора в стек

}

}

else if (strcmp(str, "(") == 0) // проверка на равенство строки левой скобке

{

stack\_push(pts, str); // добавление левой скобки в стек

}

else if (strcmp(str, ")") == 0) // проверка на равенство строки правой скобке

{

while (!(strcmp(stack\_peek(pts), "(") == 0)) // цикл пока вершина не левая скобка

{

queue\_push(ptq, stack\_pop(pts)); // перенос стека в очередь

}

stack\_pop(pts); // удаление левой скобки из стека

}

}

strcpy(str, ""); // обнуление строки

}

}

while (pts->top > -1) // перенос оставшихся элементов из стека в очередь

{

queue\_push(ptq, stack\_pop(pts));

}

}

int lencount(FILE \*fp, char \*filename) // подсчет длины файла

{

fp = fopen(filename, "r"); // открытие файла

int count = 0; // переменная для длины строки

while (!feof(fp)) // цикл для подсчета длины строки

{

char ch = fgetc(fp); // чтение файла посимвольно

count++; // увеличение длины

}

fclose(fp); // закрытие файла

return count;

}

char \*trueform(FILE \*fp, char \*filename)

{

fp = fopen(filename, "r"); // открытие файла

char str[lencount(fp, filename)]; // создание строки длиной аналогичной строке из файла

fgets(str, sizeof(str), fp); // извлечение строки из файла

char \*truestr = (char \*)malloc(sizeof(str) \* 2 \* sizeof(char)); // строка для форматирования

int j = 0; // переменная счетчик

for (int i = 0; i < sizeof(str); i++) // цикл форматирования

{

if (str[i + 1] == ',') // проверка ch на равенство точке

{

str[i + 1] = '.'; // замена ch на запятой

}

if ((isdigit(str[i]) || str[i] == '.') & (isdigit(str[i + 1]) || str[i + 1] == '.')) // если символ - число и после него число

{

truestr[j] = str[i]; // добавление в отформатированную строку символа

j++; // увеличение счетчика

}

else if (str[i] != ' ' & str[i + 1] != ' ') // если символ не равен пробелу и после него нет пробела

{

truestr[j] = str[i]; // добавление в отформатированную строку символа

truestr[j + 1] = ' '; // добавление в отформатированную строку пробела после символа

j += 2; // увеличение счетчика на 2

}

else // остальные случаи

{

truestr[j] = str[i]; // добавление в отформатированную строку символа

j++; // увеличение счетчика

}

}

return truestr;

}

int main(int argc, char \*argv[])

{

if (argc == 2) // проверка количества аргументов

{

FILE \*fp; // создание поинтера на файл

char \*sum = trueform(fp, argv[1]); // построение строки из файла

struct stack \*pts = newStack(strlen(sum)); // создание stack с вычислением емкости

struct queue \*ptq = newQueue(strlen(sum)); // создание queue с вычислением емкости

polish(pts, ptq, sum); // вычисление выражения

while (ptq->amount > 0) // цикл для вывода результата

{

printf("%s ", queue\_pop(ptq)); // извлечения нижнего элемента из stack

}

puts("");

}

return 0;

}

## 6.3 Пример работы программы

Содержимое текстового файла test.txt (рис.12):

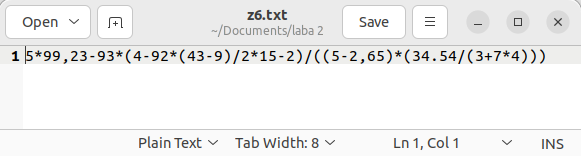
5\*99,23-93\*(4-92\*(43-9)/2\*15-2)/((5-2,65)\*(34.54/(3+7\*4)))

Рисунок 12. Текст к упражнению 6

На рис. 12 изображена работа программы для упражнения 6.



Рисунок 12. Пример работы программы упражнения 4

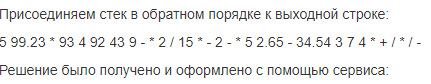


Рисунок 12. Проверка результата с помощью https://www.semestr.online/