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 Братчиков З.С.

Задание 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.

Решение:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

long int randomGenerator(void) {

return rand() % 100; // Генерация случайного числа в диапазоне [0, 99]

}

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

// Проверка наличия аргумента командной строки

if (argc != 2) {

fprintf(stderr, "Использование: %s <название\_файла>\n", argv[0]);

return 1;

}

// Инициализация генератора случайных чисел

srand(time(NULL));

// Запрос у пользователя ввода размера массива

int size;

printf("Введите размер массива: ");

scanf("%d", &size);

// Создание и заполнение массива случайными числами

int \*array = malloc(size \* sizeof(int));

if (array == NULL) {

fprintf(stderr, "Ошибка выделения памяти\n");

return 1;

}

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

array[i] = randomGenerator();

printf("array[%d]=%d\n", i, array[i]);

}

// Открытие файла для записи в обратном порядке

FILE \*file = fopen(argv[1], "w");

if (file == NULL) {

fprintf(stderr, "Ошибка открытия файла\n");

free(array);

return 1;

}

// Запись элементов массива в обратном порядке в файл

for (int i = size - 1; i >= 0; i--) {

fprintf(file, "array[%d]=%d\n", i, array[i]);

}

// Закрытие файла и освобождение выделенной памяти

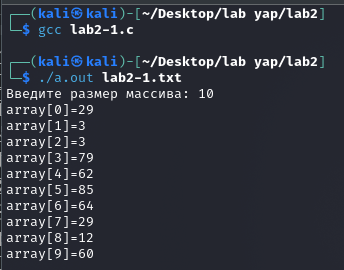
fclose(file);

free(array);

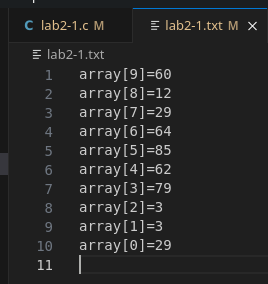
return 0;

}

Вывод программы в терминале:



Файл после работы программы:



Задание 2

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.

Решение:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Функция для обмена элементов в сортировке

void swap(double \*xp, double \*yp)

{

double temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// Функция пузырьковой сортировки

void bubbleSort(double arr[], int n)

{

int i, j;

for (i = 0; i < n - 1; i++)

{

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

{

if (arr[j] > arr[j + 1])

{

// Вызов функции для обмена элементов

swap(&arr[j], &arr[j + 1]);

}

}

}

}

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

{

// Проверка на правильное количество аргументов командной строки

if (argc != 2)

{

printf("Используйте: %s <имя\_файла>\n", argv[0]);

return 1;

}

printf("Введите размер массива:\n");

int sizeMas;

scanf("%d", &sizeMas);

// Объявление массива чисел с плавающей точкой

double mas[sizeMas];

printf("Массив со случайными числами:\n");

srand(time(NULL));

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

{

// Заполнение массива случайными числами от 0 до 1

mas[i] = (double)rand() / RAND\_MAX;

printf("mas[%d]=%f\n", i, mas[i]);

}

// Вызов функции сортировки пузырьком

bubbleSort(mas, sizeMas);

// Открытие файла для записи

FILE \*file;

file = fopen(argv[1], "w");

if (file == NULL)

{

// Вывод сообщения об ошибке, если файл не удалось открыть

printf("Не удалось открыть файл для записи.\n");

return 1;

}

// Запись отсортированного массива в файл

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

{

fprintf(file, "mas[%d]=%f\n", i, mas[i]);

}

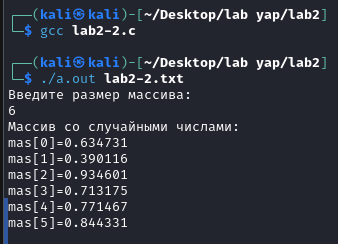
// Закрытие файла

fclose(file);

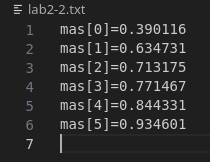
return 0;

}

Вывод программы в терминале:



Файл после работы программы:



Задание 3

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.

Решение:

#include <stdio.h>

struct goods

{

char name[128];

float price;

};

// Функция для обмена элементов в сортировке

void swap(struct goods \*xp, struct goods \*yp)

{

struct goods temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// Функция пузырьковой сортировки

void bubbleSort(struct goods \*mas, int countString)

{

int i, j;

for (i = 0; i < countString - 1; i++)

{

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

{

if (mas[j].price > mas[j + 1].price)

{

swap(&mas[j], &mas[j + 1]);

}

}

}

}

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

{

FILE \*inputFile;

FILE \*outputFile;

// Открываем входной файл для чтения

inputFile = fopen(argv[1], "r");

// Проверка на успешное открытие файла

if (inputFile == NULL)

{

printf("Ошибка при открытии входного файла.\n");

return 1;

}

int countString = 0;

char ch;

// Подсчет количества строк в файле

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

{

if (ch == '\n')

{

countString++;

}

}

// Создание массива структур для хранения товаров

struct goods shop[countString];

int i = 0;

// Сброс указателя в начало файла

fseek(inputFile, 0, SEEK\_SET);

// Заполнение массива данными из файла

while (i < countString && fscanf(inputFile, "%127s %f", shop[i].name, &shop[i].price) == 2)

{

i++;

}

// Закрываем входной файл

fclose(inputFile);

// Сортировка массива товаров по цене

bubbleSort(shop, countString);

// Открываем выходной файл для записи

outputFile = fopen(argv[2], "w");

// Проверка на успешное открытие выходного файла

if (outputFile == NULL)

{

printf("Ошибка при открытии выходного файла.\n");

return 1;

}

// Запись отсортированных данных в выходной файл

for (int c = 0; c < countString; c++)

{

fprintf(outputFile, "%s %.2f\n", shop[c].name, shop[c].price);

}

// Закрываем выходной файл

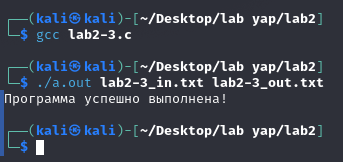
fclose(outputFile);

printf("Программа успешно выполнена!\n");

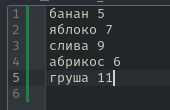
return 0;

}

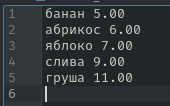
Вывод программы в терминале:



Файл с входными данными:



Файл после работы программы:



Задание 4

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.

Решение:

#include <stdio.h>

#include <stdlib.h>

#define max\_size 3

struct Node {

int data;

struct Node\* prev;

};

struct Stack {

struct Node\* top;

int count;

};

// Инициализация элемента стека

struct Node\* init\_Node(int data) {

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

if (newNode == NULL) {

printf("Ошибка при выделении памяти для узла стека.\n");

exit(1);

}

newNode->data = data;

newNode->prev = NULL;

return newNode;

}

// Инициализация стека

struct Stack\* init\_Stack() {

struct Stack\* newStack = (struct Stack\*)malloc(sizeof(struct Stack));

if (newStack == NULL) {

printf("Ошибка при выделении памяти для стека.\n");

exit(1);

}

newStack->top = NULL;

newStack->count = 0;

return newStack;

}

// Добавление нового элемента

void push(struct Stack\* stack, int data) {

struct Node\* new\_Node = init\_Node(data);

if (new\_Node == NULL) {

printf("Ошибка. Переполнение стека\n");

exit(1);

}

if (stack->count < max\_size) {

new\_Node->prev = stack->top;

stack->top = new\_Node;

stack->count++;

}

else {

printf("Стек переполнен, невозможно внести значение: %d\n", data);

free(new\_Node); // Освобождаем память, выделенную для нового узла

}

}

// Извлечение значения из стека

int pop(struct Stack\* stack) {

if (stack->top == NULL) {

printf("Ошибка. Пустой стек\n");

exit(1);

}

struct Node\* temp = stack->top;

int data = temp->data;

stack->top = temp->prev;

stack->count--;

free(temp);

return data;

}

// Просмотр верхнего значения

int peek(struct Stack\* stack) {

if (stack->top == NULL) {

printf("Ошибка. Пустой стек\n");

exit(1);

}

return stack->top->data;

}

int main() {

struct Stack\* stack = init\_Stack();

push(stack, 10);

push(stack, 20);

push(stack, 30);

printf("Верхний элемент: %d\n", peek(stack));

printf("Извлеченный элемент: %d\n", pop(stack));

push(stack, 40);

push(stack, 50);

printf("Стек:\n");

while (stack->top != NULL) {

printf("%d\n", pop(stack));

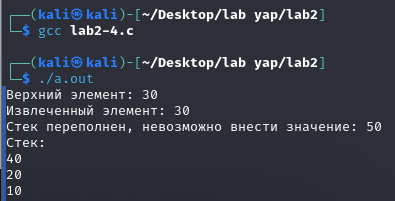
}

free(stack);

return 0;

}

Вывод программы в терминале:



Задание 5

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.

Решение:

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

struct Node {

double data;

struct Node\* prev;

};

struct Stack {

struct Node\* top;

int count;

};

// Инициализация элемента стека

struct Node\* init\_Node(double data) {

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

if (newNode == NULL) {

printf("Ошибка.\n");

exit(1);

}

newNode->data = data;

newNode->prev = NULL;

return newNode;

}

// Инициализация стека

struct Stack\* init\_Stack() {

struct Stack\* newStack = (struct Stack\*)malloc(sizeof(struct Stack));

if (newStack == NULL) {

printf("Ошибка. \n");

exit(1);

}

newStack->top = NULL;

newStack->count = 0;

return newStack;

}

// Добавление нового элемента

void push(struct Stack\* stack, double data, int max\_size) {

struct Node\* new\_Node = init\_Node(data);

if (new\_Node == NULL) {

printf("Ошибка. Переполнение стека\n");

exit(1);

}

if (stack->count < max\_size) {

new\_Node->prev = stack->top;

stack->top = new\_Node;

stack->count++;

} else {

printf("Стек переполнен, невозможно внести значение: %f\n", data);

free(new\_Node);

}

}

// Извлечение значения из стека

double pop(struct Stack\* stack) {

if (stack->top == NULL) {

printf("Ошибка. Пустой стек\n");

exit(1);

}

struct Node\* temp = stack->top;

double data = temp->data;

stack->top = temp->prev;

stack->count--;

free(temp);

return data;

}

// Просмотр верхнего значения

double peek(struct Stack\* stack) {

if (stack->top == NULL) {

printf("Ошибка. Пустой стек\n");

exit(1);

}

return stack->top->data;

}

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

if (argc != 2) {

printf("Использование: %s <имя\_файла>\n", argv[0]);

return 1;

}

FILE\* file = fopen(argv[1], "r"); // Открытие файла для чтения

if (file == NULL) {

printf("Не удалось открыть файл: %s\n", argv[1]);

return 1;

}

struct Stack\* stack = init\_Stack();

char ch;

int max\_size = sizeof(file);

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

if (isdigit(ch)) {

push(stack, ch - '0', max\_size);

} else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/') {

double el\_2 = pop(stack);

double el\_1 = pop(stack);

switch (ch) {

case '+':

push(stack, el\_1 + el\_2, max\_size);

break;

case '-':

push(stack, el\_1 - el\_2, max\_size);

break;

case '\*':

push(stack, el\_1 \* el\_2, max\_size);

break;

case '/':

push(stack, el\_1 / el\_2, max\_size);

break;

}

}

}

fclose(file);

printf("Результат: %f\n", pop(stack));

free(stack);

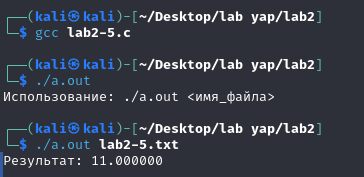
return 0;

}

Содержимое файла:



Вывод программы в терминале:



Задание 6

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 право рекурсивной
* реализовать предиктивный анализатор по составленной грамматике

Схема трансляции:

expr() -> term() exprPrime()

exprPrime() -> '+' term() exprPrime() | '-' term() exprPrime() | ε

term() -> factor() termPrime()

termPrime() -> '\*' factor() termPrime() | '/' factor() termPrime() | ε

factor() -> digit | '(' expr ')'

Где:

**expr():** Начальная точка анализа, вызывает term() и exprPrime().

**exprPrime():** Обрабатывает операторы сложения и вычитания, вызывает term() и exprPrime() в случае обнаружения соответствующего оператора.

**term():** Обрабатывает множители, вызывает factor() и termPrime().

**termPrime():** Обрабатывает операторы умножения и деления, вызывает factor() и termPrime() в случае обнаружения соответствующего оператора.

**factor():** Обрабатывает числа и выражения в скобках.

Символ ε обозначает пустую последовательность. Эта схема показывает порядок вызова функций во время разбора арифметического выражения и построения постфиксной записи.

Решение:

#include <stdio.h>

#include <ctype.h>

// Глобальные переменные

const *char*\* input; // Входная строка

*char* lookahead; // Текущий символ

// Прототипы функций

*void* expr();

*void* exprPrime();

*void* term();

*void* termPrime();

*void* factor();

*void* match(*char* *expected*);

*void* error();

// Функция для печати символа

*void* print(*char* *c*) {

printf("%c", *c*);

}

// Функция для получения следующего символа

*void* getNextToken() {

lookahead = \*input++;

}

// Функция для сопоставления текущего символа с ожидаемым

*void* match(*char* *expected*) {

if (lookahead == *expected*) {

getNextToken();

} else {

error(); // В случае несоответствия вызываем функцию ошибки

}

}

// Функция для обработки ошибок

*void* error() {

printf("Error\n");

exit(1);

}

// Начальная точка программы

*int* main() {

printf("Введите инфиксное выражение: ");

*char* buffer[100];

fgets(buffer, sizeof(buffer), stdin);

input = buffer;

getNextToken(); // Инициализация lookahead

expr(); // Начало анализа

printf("\n");

return 0;

}

// Функции для разбора грамматики

*void* expr() {

term();

exprPrime();

}

*void* exprPrime() {

if (lookahead == '+') {

match('+');

term();

exprPrime();

print('+'); // Печать оператора в постфиксной записи

} else if (lookahead == '-') {

match('-');

term();

exprPrime();

print('-'); // Печать оператора в постфиксной записи

}

}

*void* term() {

factor();

termPrime();

}

*void* termPrime() {

if (lookahead == '\*') {

match('\*');

factor();

termPrime();

print('\*'); // Печать оператора в постфиксной записи

} else if (lookahead == '/') {

match('/');

factor();

termPrime();

print('/'); // Печать оператора в постфиксной записи

}

}

*void* factor() {

if (isdigit(lookahead)) {

print(lookahead); // Печать цифры в постфиксной записи

match(lookahead);

} else if (lookahead == '(') {

match('(');

expr();

match(')');

} else {

error(); // Если символ не цифра и не открывающая скобка, вызываем ошибку

}

}

Вывод программы в терминале:

