Балтийский государственный технический университет

«ВОЕНМЕХ» им. Д. Ф. Устинова

Кафедра О7 «Информационные системы и программная инженерия»

Практическая работа №3

по дисциплине «Информатика: Основы программирования» на тему «Вспомогательные алгоритмы. Функции.

Создание статических библиотек»

Выполнил:

Студент *Фамилия И.О.*

Группа *Номер\_группы*

Преподаватель:

*Фамилия И.О.*

Санкт-Петербург 20*23* г.

# Задание 1. Восходящее программирование

## Перепишите программу 7 из ПР\_2, разделив ее текст на функции. Каждая функция должна выполнять только одно законченное действие (ввод массива, вывод массива, поиск значения в массиве, удаление элемента из массива и т.п.).

Текст вариативной части задания 7 ПР\_2.

## Исходные данные:

Переменная n размер массива, тип int.

Переменная \*m указатель на одномерный динамический массив тип int.

Переменная \*k указатель на найденный элемент массива по ключу, тип int.

Переменная key ключ, тип int.

## Результирующие данные:

*Измененный исходный целочисленный динамический массив.*

. Схема программы без разделения на функции: *(из ПР\_2)*

Здесь должна быть схема программы в соответствии с ГОСТ 19.701-90.

## Текст программы без разделения на функции: *(из ПР\_2)*

Сюда добавляем текст программы 7 из ПР\_2. Шрифт Courier New или FreeMono 10 пт, междустрочный интервал одинарный.

# Декомпозиция

## Основной алгоритм:

Схема основного алгоритма в соответствии с ГОСТ 19.701-90. Схема может быть построена любым способом, в том числе начерчена вручную на листе бумаги с помощью карандаша и линейки и сфотографирована или отсканирована.

## Вспомогательные алгоритмы:

1. Алгоритм ввода массива

*Входные данные:* имя массива, размер.

*Результирующие данные:* заполненный массив Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

void add\_elements\_in\_array(int array[], int size\_array);

Параметры:

первый параметр *– адрес первого элемента массива* второй параметр *– количество элементов массива*

Возвращаемое значение – Отсутствует.

Побочный эффект – Отсутствует.

Вспомогательные переменные:

Вспомогательные переменные не требуются.

## Алгоритм вывода массива

*Входные данные:* имя массива, размер.

*Результирующие данные:* нет Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

void output\_array(int array[], int size\_array);

Параметры:

первый параметр *– адрес первого элемента массива* второй параметр *– количество элементов массива*

Возвращаемое значение отсутствует. Побочный эффект отсутствует.

Вспомогательные переменные:

Вспомогательные переменные не требуются.

1. Алгоритм *поиска элемента по ключу*

*Входные данные:* имя массива, размер, *искомый ключ*.

Результирующие данные: номер элемента в массиве или 0 в случае отсутствия искомого значения

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

int \*find\_element\_in\_array(int array[],int size\_array,int key);

Параметры:

первый параметр *– адрес первого элемента массива* второй параметр *– количество элементов массива третий параметр – искомый ключ…*

Возвращаемое значение – адрес найденного элемента или NULL в случае отсутствия искомого значения.

Побочный эффект отсутствует. Вспомогательные переменные:

Переменная \*k казатель на найденный элемент по ключу, тип int.

1. Алгоритм *удаления элемента по ключу.*

*Входные данные:* имя массива, размер, *искомый ключ*. *Результирующие данные: размер массива, измененный массив*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

int deleting\_even\_array\_elements(int array[], int size\_array,int \*k);

Параметры:

первый параметр *– адрес первого элемента массива* второй параметр *– количество элементов массива третий параметр – ключ элемента, который надо удалить.*

Возвращаемое значение – количество элементов массива. Побочный эффект – изменение значений элементов массива. Вспомогательные переменные:

Переменная flag флаг при нахождении элемента по ключу, тип int.

Переменнаяindex индекс элемента для заполнения массива новыми значениями(размер измененного массива), тип int..

## Текст программы:

## *#include <stdio.h>*

## *#include <stdlib.h>*

## *//Прототип функции заполенения одномерного массива.*

## *void add\_elements\_in\_array(int array[], int size\_array);*

## *//Прототип функция вывода одномерного массива.*

## *void output\_array(int array[], int size\_array);*

## *//Прототип функция удаления элемента одномерного массива по значению ключа*

## *int deleting\_even\_array\_elements(int array[], int size\_array,int \*k);*

## *//Прототип функции нахождение элемента в массиве по заданому ключу.*

## *int \*find\_element\_in\_array(int array[],int size\_array,int key);*

## *int main() {*

## *int n; //Размер массива.*

## *int\* m; //Указатель на массив.*

## *int \*k; //Указатель на найденный элемент массива по ключу.*

## *int key; //Ключ.*

## *//Ввод размера массива.*

## *printf("Размер массива?: ");*

## *scanf("%d", &n);*

## *if(n <= 0)*

## *{*

## *printf("Размер должен быть > 0.\n");*

## *return 0;*

## *}*

## 

## *m = (int\*)malloc(sizeof(int)\*n); //Выделение памяти под одномерный динамический массив.*

## *add\_elements\_in\_array(m,n);//Вызываем функцию заполнения одномерного массива.*

## *printf("Введите ключ(индекс) элемента, который хотите удалить");*

## *scanf("%d", &key);//Ввод ключа*

## *k = find\_element\_in\_array(m,n,key); //Вызываем функцию нахождение элемента в массиве по заданому ключу(Возвращаемое значение адрес элемента в памяти)*

## *n = deleting\_even\_array\_elements(m,n,k);//Вызываем функцию удаления элемента одномерного массива по значению ключа*

## 

## *output\_array(m,n);//Вызываем функцию вывода одномерного массива.*

## 

## *printf("\n");*

## *free(m);//Чистим память занятую под динамический одномерный массив.*

## *return 0;*

## *}*

## *//Функция заполенения одномерного массива.*

## *void add\_elements\_in\_array(int array[], int size\_array){*

## *for(int i = 0; i < size\_array; i++){*

## *if(scanf("%d", array+i) != 1){*

## *printf("Введено не число.\n");*

## *}*

## *}*

## *}*

## *//Функция вывода одномерного массива.*

## *void output\_array(int array[], int size\_array){*

## *for(int i = 0; i < size\_array; i++){*

## *printf("%d ", array[i]);*

## *}*

## *}*

## *//Функция удаления элемента одномерного массива по значению ключа*

## *int deleting\_even\_array\_elements(int array[], int size\_array,int \*k){*

## 

## *int flag = 0; //Флаг при нахождении элемента по ключу.*

## *int index = 0; //Индекс элемента для заполнения массива новыми значениями(размер измененного массива)*

## *for(int i = 0; i < size\_array; i++) {*

## *if (&array[i] == k)*

## *{*

## *flag = 1;*

## *}*

## *if(!flag)*

## *{*

## *array[index++] = array[i];*

## *flag = 0;*

## *}*

## *flag = 0;*

## *}*

## *size\_array = index;*

## *int\* \_array = array;*

## *//Перевыделение памяти массива.*

## *if(index)*

## *array = (int\*)realloc(array, sizeof(int)\*size\_array);*

## *if(!array) {*

## *printf("Память не перевыделена!\n");*

## *array = \_array;*

## *}*

## *//Возвращаем размер измененного массива.*

## *return size\_array;*

## *}*

## *//Функция нахождение элемента в массиве по заданому ключу.*

## *int \*find\_element\_in\_array(int array[],int size\_array,int key){*

## *int \*k = NULL; //указатель на найденный элемент по ключу.*

## *for(int i = 0; i < size\_array; i++)*

## *{*

## *if(i == (key))*

## *{*

## *k = &array[i];*

## *//Если ключ найден, возвращаем адрес найденного элемента в массиве.*

## *return k;*

## *}*

## *}*

## *printf("Ключ не найден\n");*

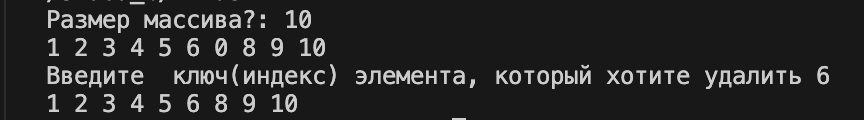
## *//Если ключ не найден, возвращаем NULL*

## *return k;*

## *}*

## Результаты работы программ: без разделения на функции *скриншот*

после разделения на функции

**

# Задание 2. Распределение памяти

* 1. *Операционная система Kubuntu 18.04.2, IDE Code::Blocks Скриншот окна работы программы*

/\* адреса функций \*/

printf("Adress of main : %p\n", main); printf("Adress of f1 : %p\n", f1); printf("Adress of f2 : %p\n", f2); printf("Adress of f3 : %p\n", f3); printf("Adress of f4 : %p\n", f4); printf("Adress of f5 : %p\n\n", f5);

Стартовые адреса функций в памяти (схемы могут быть построены с использованием приведенной таблицы, выполнены в любом удобном для Вас редакторе, а также начерчены на листе бумаги и отсканированы или сфотографированы (текст, набранный красным курсивом, нужно удалить)):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| адреса |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| функции |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Делаем выводы о размещении кода функций

/\* адрес глобальной переменной \*/

printf("var\_global : adress : %p\tvalue : %d\n\n", &var\_global, var\_global);

/\* адреса локальных переменных \*/

int var\_main = 100;

printf("var\_main : adress : %p\tvalue : %d\n", &var\_main, var\_main);

void \* pointer1\_main = malloc (1024); printf("pointer1\_main : adress : %p\t", &pointer1\_main);

printf("value : %p\tmemory block size : %d bytes\n", pointer1\_main, 1024); /\* адрес динамически выделенного блока памяти \*/

free(pointer1\_main); /\* освобождение памяти \*/

printf("pointer1\_main memory free\n\n");

/\* вызовы функций. Внутри функции f1 тоже динамически выделяется и освобождается блок памяти \*/

f1(1); f2(2,3.0);

pointer1\_main = malloc (1024); /\* повторное выделение памяти того же объема \*/

printf("pointer1\_main : new value : %p\tmemory block size : %d bytes\n\n", pointer1\_main, 1024);

/\* выделение и освобождение следующего блока памяти \*/

void \* pointer2\_main = malloc (1024);

printf("pointer2\_main : adress : %p\tvalue : %p\tmemory block size : %d bytes\n\n", &pointer2\_main, pointer2\_main, 1024); free (pointer2\_main);

printf("pointer2\_main memory free\n\n");

f1(4);

f3(5);

f4(6);

f3(7);

Распределение памяти под переменные: (схемы могут быть построены с использованием приведенной таблицы, выполнены в любом удобном для Вас редакторе, а также начерчены на листе бумаги и отсканированы или сфотографированы. Каждый столбец – один байт, адреса записывать последовательно, начиная с младшего адреса. Размещение переменных показать объединением ячеек, значения переменных записать внутри объединенной ячейки (как в заданиях 3 и 4 второй практической работы). Текст, набранный красным курсивом, нужно удалить)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| адреса |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_global |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_main |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| pointer1\_main |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| pointer\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param1\_f2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param2\_f2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| повторное выделение памяти в main | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| pointer1\_main |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| pointer2\_main |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| вызов функции f1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| pointer\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param\_f3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f4, вызов из нее f5 и рекурсивные вызовы f5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param\_f4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param\_f3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Делаем выводы о том,

* *где выделяется память под глобальные переменные,*
* *где выделяется память под локальные переменные класса auto и класса static,*
* *как выделяется память при последовательном вызове функций и в случае, когда одна функция вызывается из другой (в том числе рекурсивно из самой себя),*
* *где выделяется память при динамическом ее выделении.*
  1. *Онлайн-IDE Replit https://replit.com/languages/c Скриншот окна работы программы*

## Стартовые адреса функций в памяти:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| адреса |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| функции |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Распределение памяти под переменные: (Каждый столбец – один байт, адреса записывать последовательно, начиная с младшего адреса. Размещение переменных показать объединением ячеек, значения переменных записать внутри объединенной ячейки (как в заданиях 3 и 4 второй практической работы). Текст, набранный красным курсивом, нужно удалить)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| адреса |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_global |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_main |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| pointer1\_main |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| pointer\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param1\_f2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param2\_f2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| повторное выделение памяти в main | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| pointer1\_main |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| pointer2\_main |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| pointer\_f1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param\_f3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f4, вызов из нее f5 и рекурсивные вызовы f5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param\_f4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| param\_f5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| вызов функции f3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| param\_f3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| var\_f3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Делаем выводы о том,

* *где выделяется память под глобальные переменные,*
* *где выделяется память под локальные переменные класса auto и класса static,*
* *как выделяется память при последовательном вызове функций и в случае, когда одна функция вызывается из другой (в том числе рекурсивно из самой себя),*
* *где выделяется память при динамическом ее выделении.*

Выводы: здесь пишем общие выводы, к которым пришли при выполнении задания.

# Задание 3. Создание библиотеки функций ввода-вывода массивов

## Алгоритм ввода массива

*Входные данные:* имя массива, размер.

*Результирующие данные:* количество считанных элементов, значения элементов массива Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

Этот алгоритм описывается в программе функциями *(если одной, то пишем один прототип)*

*void add\_int\_elements\_in\_array(int \*array,int size\_array);*

*void add\_double\_elements\_in\_array(double \*array,int size\_array);*

первый параметр *– адрес первого элемента массива* второй параметр *– количество элементов массива*

Возвращаемое значение – Отсутствует.

Побочный эффект – Отсутствует.

Вспомогательные переменные:

Вспомогательные переменные не требуются.

## Алгоритм вывода массива в виде непрерывной последовательности чисел, разделенных пробелами.

*Входные данные:* имя массива, размер.

Результирующие данные: нет

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функциями

void output\_int\_array(int \*array,int size\_array);

void output\_double\_array(double \*array,int size\_array);

Параметры:

первый параметр *– адрес первого элемента массива* второй параметр *– количество элементов массива*

Возвращаемое значение отсутствует. Побочный эффект отсутствует.

Вспомогательные переменные:

Вспомогательные переменные не требуются.

1. Алгоритм вывода массива в виде таблицы. *Входные данные: имя массива, размер. Результирующие данные: нет*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функциями

void output\_double\_array(double \*array,int size\_array);

void output\_int\_array\_table(int \*array,int size\_array);)

Параметры:

первый параметр *– адрес первого элемента массива* второй параметр *– количество элементов массива*

Возвращаемое значение отсутствует. Побочный эффект отсутствует.

Вспомогательные переменные:

Вспомогательные переменные не требуются.

Заголовочный файл *3task*.h:

#include <stdio.h>

#include <stdlib.h>

//Прототип функции ввода целочисленного одномерного массива

void add\_int\_elements\_in\_array(int \*array,int size\_array);

//Прототип функции ввода вещественного одномерного массива

void add\_double\_elements\_in\_array(double \*array,int size\_array);

//Прототип функции вывода целочисленного массива в виде непрерывной последовательности чисел разделенных пробелами

void output\_int\_array(int \*array,int size\_array);

//Прототип функции вывода вещественного массива в виде непрерывной последовательности чисел разделенных пробелами

void output\_double\_array(double \*array,int size\_array);

//Прототип функции вывода целочисленного массива в виде таблицы

void output\_int\_array\_table(int \*array,int size\_array);

//Прототип функции вывода вещественного массива в виде таблицы

void output\_double\_array\_table(double \*array,int size\_array);

Файл реализации *3\_task*.c

*#include <stdio.h>*

*#include <stdlib.h>*

*//Функция ввода целочисленного одномерного массива*

*void add\_int\_elements\_in\_array(int \*array,int size\_array){*

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

*scanf("%d", array+i);*

*}*

*}*

*//Функция ввода вещественного одномерного массива*

*void add\_double\_elements\_in\_array(double \*array,int size\_array){*

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

*scanf("%lf", array+i);*

*}*

*}*

*//Функция вывода целочисленного массива в виде непрерывной последовательности чисел разделенных пробелами*

*void output\_int\_array(int \*array,int size\_array){*

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

*printf("%d ", \*(array+i));*

*}*

*}*

*//Функция вывода вещественного массива в виде непрерывной последовательности чисел разделенных пробелами*

*void output\_double\_array(double \*array,int size\_array){*

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

*printf("%lf ", \*(array+i));*

*}*

*}*

*//Функция вывода целочисленного массива в виде таблицы*

*void output\_int\_array\_table(int \*array,int size\_array){*

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

*printf("|%5d| ", \*(array+i));*

*}*

*}*

*//Функция вывода вещественного массива в виде таблицы*

*void output\_double\_array\_table(double \*array,int size\_array){*

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

*printf("|%5lf| ", \*(array+i));*

*}*

*}*

Полученный файл библиотеки *имя\_файла*.*a* имеет размер *579* байт.

# Задание 4. Создание библиотеки для работы с динамической матрицей

## Алгоритм выделения памяти под прямоугольную динамическую матрицу

Входные данные: количество блоков данных (строк или столбцов), количество элементов в одном блоке (строке или столбце), размер одного элемента данных в байтах.

Результирующие данные: адрес начала блока указателей

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

*void\*\* init\_dynamic\_matrix(int row, int column, int size\_of\_data);*

первый параметр *– Кол-во строк в матрице.*

второй параметр *– Кол-во столбцов в матрице.*

третий параметр *–выделяемая память под 1 элемент.*

Возвращаемое значение – адрес начала блока указателей или NULL в случае невозможности выделить память

Побочный эффект отсутствует

* 1. Алгоритм освобождения памяти от динамической матрицы. *Входные данные: адрес начала блока указателей, количество блоков данных*. *Результирующие данные: нет*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

void clear\_memmory\_of\_matrix(void \*\*matrix,int row);

Параметры:

первый параметр *– адрес начала блока указателей* второй параметр *– кол-во строк в матрице*

Возвращаемое значение отсутствует. Побочный эффект отсутствует.

* 1. Алгоритм *Заполнения целочисленной матрицы.*

*Входные данные: адрес начала блока указателей, количество строк, кол-во столбцов.*

*Результирующие данные: Заполненная матрица.*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

## void add\_int\_elements\_in\_matrix(int \*\*matrix,int row,int column);

Параметры:

первый параметр *– адрес начала блока указателей* второй параметр *– кол-во строк в матрице*

*третий параметр – кол-во столбцов в матрице*

Возвращаемое значение отсутствует. Побочный эффект отсутствует.

4)Алгоритм *Заполнения вещественной матрицы.*

*Входные данные: адрес начала блока указателей, количество строк, кол-во столбцов.*

*Результирующие данные: Заполненная матрица.*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

## void add\_double\_elements\_in\_matrix(double \*\*matrix,int row,int column);

Параметры:

первый параметр *– адрес начала блока указателей* второй параметр *– кол-во строк в матрице*

*третий параметр – кол-во столбцов в матрице*

Возвращаемое значение отсутствует. Побочный эффект отсутствует.

5)Алгоритм *Вывода целочисленной матрицы.*

*Входные данные: адрес начала блока указателей, количество строк, кол-во столбцов.*

*Результирующие данные: Матрица*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

## void output\_int\_matrix(int \*\*matrix,int row,int column);

Параметры:

первый параметр *– адрес начала блока указателей* второй параметр *– кол-во строк в матрице*

*третий параметр – кол-во столбцов в матрице*

Возвращаемое значение отсутствует. Побочный эффект отсутствует.

6)Алгоритм *Вывода вещественной матрицы.*

*Входные данные: адрес начала блока указателей, количество строк, кол-во столбцов.*

*Результирующие данные: Матрица*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

## void output\_double\_matrix(double \*\*matrix,int row,int column, int numbers\_after\_dot);

Параметры:

первый параметр *– адрес начала блока указателей* второй параметр *– кол-во строк в матрице*

*третий параметр – кол-во столбцов в матрице*

*четвертый параметр – кол-во знаков после запятой.*

Возвращаемое значение отсутствует. Побочный эффект отсутствует.

Заголовочный файл *4task*.h:

*//Прототип функции выделения памяти под динамическую матрицу*

*void\*\* init\_dynamic\_matrix(int row, int column, int size\_of\_data);*

*//Прототип функции отчистки памяти динамической матрицы*

*void clear\_memmory\_of\_matrix(void \*\*matrix,int row);*

*//Прототип функции заполнения целочисленной динамической матрицы*

*void add\_int\_elements\_in\_matrix(int \*\*matrix,int row,int column);*

*//Прототип функции заполнения вещественной динамической матрицы*

*void add\_double\_elements\_in\_matrix(double \*\*matrix,int row,int column);*

*//Прототип функции вывода целочисленной матрицы*

*void output\_int\_matrix(int \*\*matrix,int row,int column);*

*//Прототип функции вывода вещественной матрицы*

*void output\_double\_matrix(double \*\*matrix,int row,int column, int numbers\_after\_dot);*

Файл реализации*4\_task*.c

#include <stdlib.h>

#include <stdio.h>

#include "3task.h"

//Функция выделения памяти под динамическую матрицу

void\*\* init\_dynamic\_matrix(int row, int column, int size\_of\_data){

void \*\*m;

m = (void\*\*) malloc(row \* size\_of\_data);

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

{

m[i] =(void\*) malloc( column \* size\_of\_data);

}

if(!m){

printf("Память не выделена\n");

return NULL;

}

//printf("Адрес блока начала указателей: %p\n",m);

return m;

}

//Функция отчистки памяти динамической матрицы

void clear\_memmory\_of\_matrix(void \*\*matrix,int row){

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

free(matrix[i]);

}

//Функция заполнения целочисленной динамической матрицы

void add\_int\_elements\_in\_matrix(int \*\*matrix,int row,int column){

//заполняем матрицу по строчно

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

{

add\_int\_elements\_in\_array(\*(matrix+i),column);//Вызываем функию заполенния одномерного массива

}

}

//Функция заполнения вещественной динамической матрицы

void add\_double\_elements\_in\_matrix(double \*\*matrix,int row,int column){

//заполняем матрицу по строчно

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

{

add\_double\_elements\_in\_array(\*(matrix+i),column);//Вызываем функию заполенния одномерного массива

}

}

//Функция вывода целочисленной матрицы

void output\_int\_matrix(int \*\*matrix,int row,int column){

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

{

for(int j=0;j<column;j++)

{

printf("%5d",matrix[i][j]);

}

printf("\n");

}

}

//Функция вывода вещественной матрицы

void output\_double\_matrix(double \*\*matrix,int row,int column, int numbers\_after\_dot){

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

{

for(int j=0;j<column;j++)

{

printf("%lf ",matrix[i][j]);

//printf("%.d.%lf",matrix[i][j]);

}

printf("\n");

}

}

Полученный файл библиотеки *имя\_файла*.*a* имеет размер *579* байт.

# Задание 5. Обработка одномерных и многомерных массивов

* 1. Даны два вектора Y и X, содержащих по n элементов каждый. Вычислить значениефункции

## Исходные данные:

Переменная n кол-во элементов в векторах, тип int.

Переменная \*x указатель на вещественный массив X, тип double.

Переменная \*y указатель на вещественный массив Y, тип double.

Переменная chislit значение числителя, тип double.

Переменная znamenat значение знаменателя, тип double.

## Результирующие данные:

*Переменная answer числовое значение функции, тип double.*

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| *наборы данных должны соответствовать максимально большему количеству возможных вариантов* |  | *скриншот* |
|  |  | *скриншот* |
|  |  | *скриншот* |
|  |  | *скриншот* |

## Основной алгоритм:

Схема основного алгоритма в соответствии с ГОСТ 19.701-90. Схема может быть построена любым способом, в том числе начерчена вручную на листе бумаги с помощью карандаша и линейки и сфотографирована или отсканирована.

## Вспомогательные алгоритмы:

1. *Для ввода элементов массива используется функция add\_double\_elements\_in\_array из созданной библиотеки 4task.h*
2. Алгоритм *вычисления числителя функции*

*Входные данные:* аргумент функции.

Результирующие данные: числовое значение числителя.

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

int count\_chislit(double \*x, double\*y,int n);

Параметры:

первый параметр *– адрес первого элемента массива1* второй параметр *– адрес первого элемента массива2*

третий параметр *– кол-во элементов в массиве*

Возвращаемое значение – Значение числителя

Побочный эффект отсутствует.

3)Алгоритм *Вычисления знаменателя функции.*

*Входные данные:* адрес первого элемента массива, кол-во элементов в массиве.

Результирующие данные: числовое значение знаменателя.

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

int count\_znamenat(double \*x, double \*y, int n);

Параметры:

первый параметр *– адрес первого элемента массива1* второй параметр *– адрес первого элемента массива2*

третий параметр *– кол-во элементов в массиве*

Возвращаемое значение – Значение знаменателя

Побочный эффект отсутствует.

4)Алгоритм *Вычисления суммы элементов вектора.*

*Входные данные:* адрес первого элемента массива, кол-во элементов в массиве.

Результирующие данные: Сумма элементов.

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

int sum\_of\_vector(int \*vector, int n);

Параметры:

первый параметр *– адрес первого элемента массива*

второй параметр *– кол-во элементов*

Возвращаемое значение – Значение суммы

Побочный эффект отсутствует.

## Текст программы:

#include <stdio.h>

#include <stdlib.h>

#include "4task.h"

#include <math.h> //Импортируем нашу библиотеку для работы с матрицами и строками

//Прототип функции суммы элементов вектора

int sum\_of\_vector(int \*vector, int n);

//Прототип функции числителя

int count\_chislit(double \*x, double\*y,int n);

//Прототип функции подсчета знаменателя

int count\_znamenat(double \*x, double \*y, int n);

int main(){

int n; //Кол-во элементов у вектора.

double \*x,\*y; //Указатели на вещественные массивы.

double chislit,znamenat; //Значения числителя и знаменателя.

double answer; //Значения функции.

scanf("%d", &n);

add\_double\_elements\_in\_array(x,n); //Вызываем функцию заполнения массива

add\_double\_elements\_in\_array(y,n); //Вызываем функцию заполнения массива

chislit = count\_chislit(x,y,n); //Вызываем функцию подсчета числителя функции(Возвращаемое значение - численное значение числителя)

znamenat = count\_znamenat(x,y,n); //Вызываем функцию подсчета знаменателя функции((Возвращаемое значение - численное значение знаменателя))

answer = chislit/ znamenat;

printf("%lf\n", answer);

return 0;

}

//Функция суммы элементов вектора

int sum\_of\_vector(int \*vector, int n){

int sum = 0;

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

sum += vector[i];

}

return sum;

}

//Функция подсчета числителя

int count\_chislit(double \*x, double\*y,int n){

double sumX = 0; //Сумма всех элементов веткора x

double sumY = 0; //Сумма всех элементов ветктора Y

double all\_sum = 0; // Значение первого слагаемого числителя

double n\_y;

double otv = 0; //Значение числителя

sumX = sum\_of\_vector(x,n); //Вызываем функцию для подсчета суммы всех элеметов вектора X

sumY = sum\_od\_vector(y,n); //Вызываем функцию для подсчета суммы всех элеметов вектора Y

//Считаем Первое слагаемое числителя

for(int j = 0; j < n; j++){

n\_y = y[j] \*(sumX \*n);

all\_sum +=n\_y; // Первое слагамое

}

//Значение числителя

otv = (all\_sum - (sumX\*sumY));

return otv;

}

//Функция подсчета знаменателя

int count\_znamenat(double \*x, double \*y, int n){

double sumX,sumX\_2; //Значение сумм слагаемых

double sumY,sumY\_2;

double otv; //Значение знаменателя

double first,second; //Значение первого и второго множителей

//Первое слагаемое

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

sumX += (x[i])\*(x[i]);

}

//Второе слагаемое

sumX\_2 = sum\_of\_vector(x,n);//Вызываем функцию для подсчета суммы всех элеметов вектора X

sumX\_2 = sumX\_2\*sumX\_2;

//Первый множитель

first = sumX - sumX\_2;

//Первое слагаемое

for(int j = 0; j< n; j++){

sumY += (y[j])\*(y[j]);

}

//Второе слагаемое

sumY\_2 = sum\_of\_vector(x,n);//Вызываем функцию для подсчета суммы всех элеметов вектора X

sumX\_2 = sumX\_2\*sumX\_2;

//Второй множитель

second = sumY - sumY\_2;

//Численное значение знаменателя

otv = first\*second;

otv = sqrt(otv);

return otv;

}

* 1. Имеется две квадратные матрицы разных порядков. В каждой строке каждой матрицы сменить знак максимальных по модулю элементов на противоположный, если найденные элементы не лежат на диагоналях матриц. Если все числа в строке по модулю одинаковы, изменений производить так же не требуется.

## Исходные данные:

Переменная \*\*matrix указатель на целочисленную матрицу, тип int.

Переменная row кол-во строк в матрице, тип int.

Переменная column кол-во столбцов в матрице, тип int.

Переменная size\_of\_data выделяемая память под элемент матрицы, тип int.

Переменная \*\*p указатель на динамический двумерный массив, тип void.

## Результирующие данные:

*описываем выходные данные, их обозначение в программе и тип*. Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| *наборы данных должны соответствовать максимально большему количеству возможных вариантов* |  | *скриншот* |
|  |  | *скриншот* |
|  |  | *скриншот* |
|  |  | *скриншот* |

## Основной алгоритм:

Схема основного алгоритма в соответствии с ГОСТ 19.701-90. Схема может быть построена любым способом, в том числе начерчена вручную на листе бумаги с помощью карандаша и линейки и сфотографирована или отсканирована.

## Вспомогательные алгоритмы:

1. *Для ввода элементов массива используется функция имя\_функции из созданной библиотеки имя\_библиотеки (если это не так, то описываем всё полностью: входные и выходные данные, схему, функцию и т.д.)*
2. Алгоритм *поиска элемента по ключу и описывающая его функция те же, что и в предыдущей задаче (если это не так, то описываем всё полностью: входные и выходные данные, схему, функцию и т.д.)*
3. Алгоритм *такой-то*

(описываем всё полностью: входные и выходные данные, схему, функцию и т.д.)

…

и так далее для всех вспомогательных алгоритмов

## Текст программы:

Сюда добавляем текст программы **с комментариями**. Шрифт Courier New или FreeMono 10 пт, междустрочный интервал одинарный.

5.3 Подсчитать общее количество простых чисел среди элементов целочисленных массивов X (50) и Y (12х5). Простым называется натуральное число, не имеющее других делителей, кроме 1 и значения самого числа.

## Исходные данные:

Переменная size\_array размер одномерного массива, тип int.

Переменная row кол-во строк в матрице, тип int.

Переменная column кол-во столбцов в матрице, тип int.

Переменная \*x указатель на одномерный массив, тип int.

Переменная size\_data размер памяти для одного элемента массива, тип int.

Переменная \*\*p указатель на динамический двумерный массив, тип void.

Переменная simple\_array кол-во простых чисел у массива, тип int.

Переменная simple\_matrix кол-во простых чисел у матрицы, тип int.

## Результирующие данные:

*описываем выходные данные, их обозначение в программе и тип*. Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| *25*  *1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25*   1. *3 16*   *1 2 3*  *4 5 6*   1. *8 9* 2. *10 11 12* 3. *13 14 15* 4. *16 17 18* | 9  7 |  |
|  |  | *скриншот* |
|  |  | *скриншот* |
|  |  | *скриншот* |

1. Алгоритм *Поиска простых чисел у матрицы*

*Входные данные: адрес начала блока указателей, количество строк, кол-во столбцов.*

*Результирующие данные: кол-во простых чисел*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

## int find\_simple\_chars\_matrix(int \*\*matrix,int row, int column);

Параметры:

первый параметр *– адрес начала блока указателей* второй параметр *– кол-во строк в матрице*

*третий параметр – кол-во столбцов в матрице*

Возвращаемое значение Кол-во простых элементов в матрице.. Побочный эффект отсутствует.

3)Алгоритм *Поиска простых чисел у массив*

*Входные данные: адрес первого элемента массива, размер массива..*

*Результирующие данные: кол-во простых чисел*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

## int find\_simple\_chars\_array(int \*array,int size\_array);

Параметры:

первый параметр *– адрес первого элемента массива,*

второй параметр *– размер массива.*

Возвращаемое значение Кол-во простых элементов в массиве.. Побочный эффект отсутствует.

## Текст программы:

#include <stdio.h>

#include <stdlib.h>

#include "4task.h" //Импортируем нашу библиотеку для работы с матрицами

//Прототип функции поиска простых чисел у одномерного массива

int find\_simple\_chars\_array(int \*array,int size\_array);

//Прототип функции поиска простых чисел у матрицы

int find\_simple\_chars\_matrix(int \*\*matrix,int row, int column);

int main(){

int size\_array; //Рзамер массива

int row,column; //кол-во строк и столбцов у матрицы

int \*x; //указатель на массив

int size\_data; //память под 1 элемент матрицы

void \*\*p; //указатель на матрицу

int simple\_array,simple\_matrix=0; //Кол-во простых элементов у массива, матрицы

//Вводим размер массива

printf("Введите размер одномерного массива\n");

scanf("%d", &size\_array);

if(size\_array <= 0){

printf("Размер должен быть > 0.\n");

return 0;

}

//Выделяем память под одномерный массив

x = (int\*)malloc(sizeof(int)\*size\_array);

add\_int\_elements\_in\_array(x,size\_array); //Вызываем функцию добавления элементов в массив

printf("Ваш массив\n");

output\_int\_array(x,size\_array); //Вызываем функцию вывода массива

simple\_array = find\_simple\_chars\_array(x,size\_array); //Вызываем функию поиска простых чисел у массива(Возвращаемое значение кол-во найденых элементов)

//Вводим кол-во строк, столбцов и определяемую память

printf("\nВведите кол-во строк, столбцов и память\n");

scanf("%d %d %d", &row,&column,&size\_data);

if((row && column ) <= 0){

printf("Размер должен быть > 0.\n");

return 0;

}

p = init\_dynamic\_matrix(row,column,size\_data); //Вызываем функцию для выделения памяти под матрицу

int \*\*ptr\_matrix = (int\*\*) p; //Указатель для целочисленной матрицы

add\_int\_elements\_in\_matrix(ptr\_matrix,row,column); //Вызываем функцию добавления элементов в матрицу

printf("Ваша матрица\n");

output\_int\_matrix(ptr\_matrix,row,column); //Вызываем функцию вывода матрицы

simple\_matrix = find\_simple\_chars\_matrix(ptr\_matrix,row,column); //Вызываем функию поиска простых чисел у матрицы(Возвращаемое значение кол-во найденых элементов)

printf("Кол-во простых чисел у массива: ");

printf(" %d\n",simple\_array);

printf("Кол-во простых чисел у матрицы: ");

printf(" %d\n",simple\_matrix);

//Чистим память

free(x);

clear\_memmory\_of\_matrix(p,row);

return 0;

}

//Функция поиска простых чисел у одномерного массива

int find\_simple\_chars\_array(int \*array,int size\_array){

int count = 0; //Кол-во простых элементов в массиве

int flag = 0; //Кол-во делителей у элемента массива

//Если у элемента массива 2 делителя(само число и 1), то к счетчику простых элементов прибавляем 1

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

for(int j = 1; j <=size\_array; j++){

if ((\*(array+i))% j == 0){

flag +=1;

}

}

if (flag == 2){

count+=1;

}

flag = 0;

}

//Возвращаем кол-во простых чисел у массива

return count;

}

//Функцция посика простых элементов у матрицы

int find\_simple\_chars\_matrix(int \*\*matrix,int row, int column){

int count = 0; //Кол-во простых элементов у матрицы

int flag = 0; //Кол-во делителей у элемента матрицы

//Если у элемента матрицы 2 делителя(само число и 1), то к счетчику простых элементов прибавляем 1

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

for(int j =0; j < column; j++){

for(int k = 1; k <= (row\*column); k++){

if(matrix[i][j] % k == 0){

flag +=1;

}

}

if (flag == 2){

count+=1;

}

flag = 0;

}

}

//Возвращаем кол-во простых чисел у матрицы

return count;

}

*10.1* Некто хочет выпилить табуретку с перекладинами из имеющегося у него деревянного прямоугольного параллелепипеда. Опишите заготовку трехмерным массивом и обозначьте единицами элементы, «принадлежащие» табуретке (которые должны остаться после обработки), остальные элементы обнулите. Вывод массива на экран осуществлять послойно.

## Исходные данные:

Переменная X размер массива по глубине, тип int.

Переменная Y размер массива по ширине, тип int.

Переменная Z размер массива по высоте, тип int.

Переменная \*\*\*arr указатель на трехмерный динамический массив, тип int.

## Результирующие данные:

*Измененный трехмерный целочисленный массив в виде табуретки.*

Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| *наборы данных должны соответствовать максимально большему количеству возможных вариантов* |  | *скриншот* |
|  |  | *скриншот* |
|  |  | *скриншот* |
|  |  | *скриншот* |

1)Алгоритм *вывода трехмерного массива*

*Входные данные: адрес начала блока указателей, высота, глубина, ширина.*

*Результирующие данные: Нет.*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

## void output\_array(int \*\*\*array, int x,int y,int z);

Параметры:

первый параметр *– адрес начала блока указателей* второй параметр *– глубина массива*

*третий параметр – ширина массива*

*четвертый параметр – высота массива*

Возвращаемое значение отсутствует.

Побочный эффект отсутствует.

2)Алгоритм *выделения памяти под трехмерный массив*

*Входные данные: высота, глубина, ширина.*

*Результирующие данные: Адрес начала блока указателей .*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

## int \*\*\*init\_three\_dimensional\_rray(int X, int Y, int Z);

Параметры:

первый параметр *–Глубина*

второй параметр *– ширина*

*третий параметр – высота*

Возвращаемое значение Адрес начала блока указателей .

Побочный эффект отсутствует.

3)Алгоритм создания табуретки

*Входные данные: указатель на массив, высота, глубина, ширина.*

*Результирующие данные: Заполненный массив.*

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

void create\_stool(int \*\*\*array, int x, int y, int z);

Параметры:

первый параметр *–Адрес начала блока указателей*

второй параметр *– глубина*

*третий параметр – ширина*

*третий параметр – высота*

Возвращаемое значение отсутствует.

Побочный эффект отсутствует.

## Текст программы:

#include <stdio.h>

#include <stdlib.h>

//Прототип функции вывода трехмерного массива

void output\_array(int \*\*\*array, int x,int y,int z);

//Прототип функции выделения памяти под трехмерную матрицу

int \*\*\*init\_three\_dimensional\_rray(int X, int Y, int Z);

//Прототип функции создания табуретки

void create\_stool(int \*\*\*array, int x, int y, int z);

//Лучше использовать со значениями 10 10 10!

int main() {

int X,Y,Z; //Размеры бруска

int \*\*\*arr; //Указатель на трехмерный массив

printf("Введите размер вашей заготовки X Y Z: ");

scanf("%d %d %d", &X, &Y, &Z);

arr = init\_three\_dimensional\_rray(X,Y,Z); //Вызываем функцию для выделения памяти под трхмерный массив

create\_stool(arr,X,Y,Z); //Вызываем функция для создания табуретки(Заполнение массива нулями)

output\_array(arr,X,Y,Z); //Вызываем функция вывода массива

return 0;

}

//Функция выделения памяти под трехмерную матрицу

int \*\*\*init\_three\_dimensional\_rray(int X, int Y, int Z){

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

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

array[i] =(int\*\*) malloc(X\* sizeof(int\*));

for(int j=0;j<X;j++){

array[i][j] = (int\*) malloc(Y\* sizeof(int));

}

}

if(!array){

printf("Память не выделена\n");

return NULL;

}

return array;

}

//Функция вывода трехмерного массива

void output\_array(int \*\*\*array, int x,int y,int z) {

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

printf("Layer %d:\n", i);

for (int j = 0; j < y; j++) {

for (int k = 0; k < x; k++) {

printf("%d", array[i][j][k]);

}

printf("\n");

}

printf("\n");

}

}

//Функция создания табуретки

void create\_stool(int \*\*\*array, int x, int y, int z) {

//Заполняем массив нулями

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

for (int j = 0; j < y; j++) {

for (int k = 0; k < x; k++) {

array[i][j][k] = 0;

}

}

}

//Определяем размеры табуретки

int layer = z/2; //Высота табуретки

int wid = x;

int hig = y;

int perekladina = z - 7; //Расположение перекладины

// Заполняем элементы, принадлежащие табуретке, единицами

int stopx = x -3;

int stopy = y -3;

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

for (int j = 0; j < hig; j++) {

for (int k = 0; k < wid; k++) {

//Заполнение ножек табуретки

if( ((i<(perekladina-1)) || ( i > (perekladina -1))) && (i<=(layer-2))){

if( ((j<2) || (j > stopx)) && ((k<2) || (k > stopy))){

array[i][j][k] = 1;

}

}

//Заполнение перекладины табуретки

if(i ==(perekladina-1)){

if(((k<2) || (k > stopy)) || ((j < 2) || (j > stopx))){

array[i][j][k] = 1;

}

}

//Заполнение сидушки табуретки

if(i == (layer-1)){

array[i][j][k] = 1;

}

}

}

}

}

# Задание 6. Вычисление интеграла

## Определить функцию *Integral()* для приближенного вычисления определенного

интеграла вида

*b*

∫ *f* ( *x* ) *dx*

*a*

методом *Ньютона* с задаваемой параметром точностью.

Использовать эту функцию для вычисления значений двух интегралов: , , передавая в функцию *Integral()* подынтегральную функцию, пределы интегрирования и точность вычислений

(), значение n должно быть кратным 3

-Формула Ньютона

## Исходные данные:

* Переменная a нижний предел интеграла, тип double.
* Переменная b верхний предел интеграла, тип double.
* Подынтегральная функция , описывается функций fisrt\_function.
* Подынтегральная функция описывается функций second\_function.
* точность вычислений задается пользователем, используем для нее переменную eps типа double.

Результирующие данные:

значение интеграла – вещественное число Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| 1  ∫ *xdx*  0 | 0.5 | *скриншот* |
| 1  ∫(−*x*) *dx*  0 | -0.5 | *скриншот* |

Вспомогательные переменные:

Вспомогательные переменные не требуются.

## Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
|  |  | *скриншот* |
|  |  |  |
|  |  |  |

Схема программы

Здесь должна быть схема программы (основной алгоритм, вспомогательные алгоритмы) в соответствии с ГОСТ 19.701-90. Схема может быть построена любым способом, в том числе начерчена вручную на листе бумаги с помощью карандаша и линейки и сфотографирована или отсканирована.

## Текст программы

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

//ФПрототип функции для вычисления интеграла

void integral(double f(double), int n, int b, int a);

//Пототип функции первой функции для интегрирования

double first\_function(double x);

//Прототип функция пвторой функции для интегрирования

double second\_function(double x);

int main(){

int N;

double b,a; //Верхний и нижний предел интеграла

//Вводим n кратное 3

printf("Введите n: ");

scanf("%d", &N);

if(N%3 != 0){

printf("Введите число делящиеся на 3\n");

return 1;

}

//Обработка первого интеграла

a = 4;

b = -1;

printf("Значение 1 интеграла: ");

integral(first\_function,N,b,a); //Вызывваем функцию подсчета интеграла, предавая в него функцию first\_function(Первая функция)

printf("\n");

//Обработка второго интеграла

a = 3.5;

b = 0.5;

printf("Значение 2 интеграла: ");

integral(second\_function,N,b,a);//Вызывваем функцию подсчета интеграла, предавая в него функцию second\_function(Вторая функция)

return 0;

}

//Функция для вычисления интеграла

void integral(double f(double), int n, int b, int a){

double dx;

double count; //Счетчик для цикла

double \*X\_N; //Указатель на массив x\_0, x\_1 и тд

double \*Y\_N; //Указатель на массив y\_0, y\_1 и тд

double x\_n; //Значения на отрезках x

double sum = 0; //Сумма значений функции на каждом отрезке x

double res; //Результат интегрирования функции

double func; //Значение функции на каждом отрзке x

x\_n = a; //начальный элемент отрезков x

dx = (double) (3\*(b - a))/(8\*n); //вычисляем dx

count = (double) (b-a)/dx; //Вычисляем счетки для цикла

count +=1;

//воздаем динамические масссивы для хранения точек x и значений функций y

X\_N = malloc(count \* sizeof(double));

Y\_N = malloc(count\* sizeof(double));

//разбивание на равные части x\_0, x\_1, x\_n

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

X\_N[i] = x\_n;

x\_n += dx;

}

//нахождение значений функции на каждом отрезке y\_0, y\_1, y\_n

for(int j = 0; j < count; j++)

{

if( (j!=0) && (j!=(count-1)))

{

if(j%3 == 0){

func = 2\*f(X\_N[j]);

Y\_N[j] = func;

}

else{

func = 3\*f(X\_N[j]);

Y\_N[j] = func;

}

}

else{

func = f(X\_N[j]);

Y\_N[j] = func;

}

}

//sum += (Y\_N[0] + \*(Y\_N + ((int) count-1)))/2; //получем первое слагаемое - (y\_0 + y\_n)/2

//Получаеам оставшиеся слагаемые

for(int j = 1; j < count-1; j++){

sum += Y\_N[j];

}

res = dx\*(sum); //Вычисляем конечный результат

printf("%lf\n",res);

free(X\_N); // Чистим память занимаемую массивами

free(Y\_N);

}

// Первая функция для интегрирования

double first\_function(double x){

double f;

f = (2\*x)\*(x\*x+1);

//Возвращаем значение функции

return f;

}

// Вторая функция функция для интегрирования

double second\_function(double x){

double f;

f = (exp(2\*x))/2\*x;

//Возвращаем значение функции

return f;

}

# Задание 7. Сортировка

## Напишите программу упорядочения прямоугольной целочисленной матрицы в соответствии с указанным в вариативной части задания правилом.

## *Переставить строки прямоугольной целочисленной матрицы по возрастанию сумм*

## *элементов строки, значения которых нечетны.*

## Исходные данные:

Переменная \*\*matrix указатель на двумерный динамический массив, тип int.

Переменная row кол-во строк в матрице, тип int.

Переменная column кол-во столбцов в матрице, тип int.

Переменная size\_data память под 1 элемент массива, тип int.

## Результирующие данные:

Измененная исходная целочисленная матрица

а) Для сортировки определяется отдельная функция, *реализующая метод пузырька*. Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| *наборы данных должны соответствовать максимально большему количеству возможных вариантов* |  | *скриншот* |

|  |  |  |
| --- | --- | --- |
|  |  | *скриншот* |
|  |  | *скриншот* |
|  |  | *скриншот* |

## Основной алгоритм:

Схема основного алгоритма в соответствии с ГОСТ 19.701-90. Схема может быть построена любым способом, в том числе начерчена вручную на листе бумаги с помощью карандаша и линейки и сфотографирована или отсканирована.

## Вспомогательные алгоритмы:

1. *Для выделения памяти под динамическую матрицу используется функция init\_dynamic\_matrix из созданной библиотеки 4task.h*
2. *Для ввода элементов матрицы используется функция add\_int\_elements\_in\_matrix из созданной библиотеки 4task.h*
3. Алгоритм *сортировки последовательности*

Входные данные: количество чисел, последовательность чисел. Результирующие данные: измененная последовательность строк.

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

void replace\_rows(int \*\*matrix, int row,int column,int index,int index\_start);

Параметры:

первый параметр *– адрес начала блока указателей на массив*

второй параметр *– кол-во строк в матрице*

третий параметр *– кол-во столбцов в матрице*

четвертый параметр *– индекс строчки, которую надо переместить  
пятый параметр – индекс строки, на которую надо переместить*

Возвращаемое значение – нет.

Побочный эффект изменение последовательности элементов в массиве.

4)Алгоритм *поиск нечетных сумм элементов строк матрицы*

Входные данные: адрес начала блока указателей на матрицу, количестов данных

Результирующие данные: строка, которая находится не на своем месте.

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

void identifying\_inappropriate\_strings(int \*\*matrix, int row,int column);

Параметры:

первый параметр *– адрес начала блока указателей на массив*

второй параметр *– кол-во строк в матрице*

третий параметр *– кол-во столбцов в матрице*

Возвращаемое значение – нет.

Побочный эффект изменение последовательности элементов в массиве.

## Текст программы:

#include <stdio.h>

#include <stdlib.h>

#include "4task.h" //Импортируем нашу библиотеку для работы с матрицами

#include <limits.h>

//Прототипа функции перестановки строк матрицы.

void replace\_rows(int \*\*matrix, int row,int column,int index,int index\_start);

//Прототипа функции определения сумм строк с нечетными элементами

void identifying\_inappropriate\_strings(int \*\*matrix, int row,int column);

int main(void){

void \*\*matrix; //Указатель на двумерный динамический массив

int row,column; //кол-во строк, столбцов

int size\_data; //Память под 1 элемент двумерного массива

//Вводим кол-во строк, столбцов и память под 1 элемент двумерного массива

printf("Введите размер матрицы и память: ");

scanf("%d %d %d",&row,&column,&size\_data);

matrix = init\_dynamic\_matrix(row,column,size\_data); //Вызываем функцию для выделения памяти под динамический двумерный массив

int \*\*m = (int\*\*)matrix; //Указатель на целочисленную матрицу

printf("Заполните матрицу [%d]x[%d]\n", row,column);

add\_int\_elements\_in\_matrix(m,row,column); //Вызываем функцию заполения матрицы

printf("Ваша матрица:\n");

output\_int\_matrix(m,row,column); //Вызываем функцию вывода матрицы

identifying\_inappropriate\_strings(m,row,column); //Вызываем функцию определения сумм строк с нечетными элементами

printf("\n");

output\_int\_matrix(m,row,column); //Вызываем функцию вывода матрицы(Выводим измененную матрицу)

return 0;

}

//Функция перестановки строк матрицы

void replace\_rows(int \*\*matrix, int row,int column,int index,int index\_start){

int buf[row]; //Вспомогательный массив для хранения переданной строчки в функцию

//Сохраняем переданную строчку в массив

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

{

buf[i] = matrix[index\_start][i];

}

// Меняем строчки местами методом пузырька

for(int j =0;j<column;j++){

matrix[index\_start][j] = matrix[index][j];

matrix[index][j] = buf[j];

}

}

//Функция определения сумм строк с нечетными элементами

void identifying\_inappropriate\_strings(int \*\*matrix, int row,int column){

int max\_sum; //Максимальная сумма нечетных элементов строк

int max\_index; //Индекс строки с максмиальной суммой нечетных элементов

//Убираем строки по этапно сверху вниз

for(int c = 0; c < row; c++){

for(int i = c; i < row; i++){

int sum = 0;

for (int j = 0; j < column; j++){

//Если элемент строки нечетный, то записываем его в сумму

if(matrix[i][j] % 2 != 0){

sum += matrix[i][j];

}

}

if(sum < max\_sum){

max\_index = i; //Запоминаем индекс текущей строки, если сумма минимальная

max\_sum = sum;

}

}

replace\_rows(matrix,row,column,max\_index,c); //Вызываем функцию перестановки строк

max\_sum = INT\_MAX;

}

}

## б) Для сортировки используется стандартная функция qsort(). Таблица тестирования:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Ожидаемый результат** | **Результат работы программы** |
| *наборы данных должны соответствовать максимально большему количеству возможных вариантов* |  | *скриншот* |
|  |  | *скриншот* |
|  |  | *скриншот* |
|  |  | *скриншот* |

Основной алгоритм:

Схема основного алгоритма не отличается от приведенной в варианте а) (если это не так, то приводим новую схему).

## Вспомогательные алгоритмы:

1. *Для выделения памяти под динамическую матрицу используется функция имя\_функции из созданной библиотеки имя\_библиотеки (если это не так, то описываем всё полностью: входные и выходные данные, схему, функцию и т.д.)*
2. *Для ввода элементов матрицы используется функция имя\_функции из созданной библиотеки имя\_библиотеки (если это не так, то описываем всё полностью: входные и выходные данные, схему, функцию и т.д.)*
3. Алгоритм *сравнения строк матрицы Входные данные: две числовые последовательности.*

Результирующие данные: значение ≤ 0, если первая строка в упорядоченной матрице должна предшествовать второй, значение >0, если вторая строка в упорядоченной матрице должна предшествовать первой.

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

прототип функции (Шрифт Courier New или FreeMono 10 пт)

Параметры:

первый параметр *– адрес первой строки матрицы*

второй параметр *– адрес второй строки матрицы*

Возвращаемое значение – целое положительное число, если вторая строка в упорядоченной матрице должна предшествовать первой, в противном случае 0 или отрицательное целое число.

Побочный эффект нет.

1. Алгоритм *вычисления суммы цифр в десятичной записи числа Входные данные: целое число.*

Результирующие данные: сумма цифр в десятичной записи этого числа.

## Схема алгоритма:

Схема алгоритма в соответствии с ГОСТ 19.701-90.

## Этот алгоритм описывается в программе функцией

прототип функции (Шрифт Courier New или FreeMono 10 пт) Параметр – анализируемое целое число

Возвращаемое значение – целое число – сумма цифр в десятичной записи анализируемого числа. Побочный эффект нет.

и так далее для всех вспомогательных алгоритмов

## Текст программы:

Сюда добавляем текст программы **с комментариями**. Шрифт Courier New или FreeMono 10 пт, междустрочный интервал одинарный.