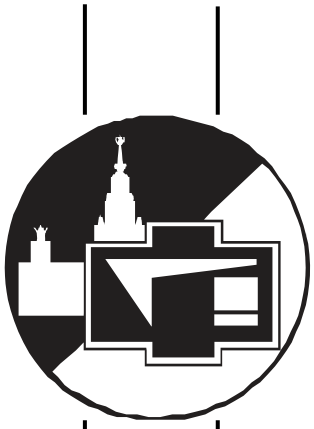

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ им. М.В. ЛОМОНОСОВА
ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

Антонюк В.А., Задорожный С.С.,Иванов А.П., Лукашёв А.А.

Практикум по программированию на языке Си

Учебно-методическое пособие (1 семестр)

Москва, 2024 г.

Антонюк В.А., Задорожный С.С., Иванов А.П., Лукашёв А.А.
Практикум по программированию на языке Си. Учебно-методическое пособие(I семестр).
М.: Физический факультет МГУ им. М.В. Ломоносова, 2024. — 100 с.

Курс «Введение в компьютерные технологии» предусматривает уверенное освоение программирования студентами первого курса. В качестве первого языка выбран один из самых распространенных (и при этом простых) языков программирования – язык программирования Си.

Учебно-методическое пособие основывается на материале, изложенном в выпущенном ранее пособии “Язык программирования Си”, и охватывает все основные разделы учебной программы курса по программированию на языке Си и подготовлено на основе курса, который много лет читается в первом семестре для студентов физического факультета МГУ. Каждая глава пособия включает задания, которые студенты должны выполнить за 4 часа практикума, и содержит пояснения, необходимые для успешного выполнения этого задания.

Для контроля успеваемости предполагается проведение коллоквиума в форме тотального опроса всех студентов по пройденному материалу. Для этого в пособии приведены возможные варианты вопросов для опроса. Также, приведены примеры дополнительных задач для студентов, успешно справившихся с заданиями.

В конце семестра проводится зачет, который в обязательном порядке включает как ответ на вопросы по теоретическому минимуму языка программирования, так и решение практической задачи, подобной тем задачам, которые студенты решали в течение семестра. В пособии приведены варианты вопросов и примеры задач.

Рассчитано на преподавателей курса «Введение в компьютерные технологии» и студентов младших курсов физико-математических специальностей.

Авторы — сотрудники кафедры математического моделирования и информатики и кафедры общей физики и волновых процессов физического факультета МГУ.

Рецензенты:
профессор физического ф-та МГУ Голубцов П.В.,
доцент физического ф-та МГУ Митин И.В.

Подписано в печать 28 октября 2024 г.Формат 60x90/16. Объём 4,72 п.л.
Тираж 50 экз. Заказ №

Отпечатано в Отделе оперативной печати физического факультета МГУ.
Физический факультет МГУ им. М.В. Ломоносова
119991 Москва, ГСП-1, Ленинские горы, д.1, стр. 2

Оглавление

Обзор курса и порядок работы студентов в практикуме

Курс «Введение в компьютерные технологии» предусматривает уверенное освоение программирования студентами первого курса. В качестве первого языка выбран один из самых распространенных (и при этом простых) языков программирования – язык программирования Си.

Вспомогательным учебным пособием по синтаксису языка Си может служить:

1. В.А. Антонюк, С.С. Задорожный, А.П. Иванов, А.А. Лукашёв, Н.А. Панов, С.А. Шленов «Язык программирования Си. Учебно-методическое пособие (I семестр).» – М.: Физический факультет МГУ им. М.В.Ломоносова, 2022, 108 с.

Оно имеется в интернете по адресу https://cmp.phys.msu.ru/sites/default/files/MetC_complete.pdf и в компьютерном классе на диске общего доступа.

На семинарах студенты получают теоретические знания по синтаксису и стандартным библиотекам этого языка, а кроме того получают базовые знания по теории вычислений, основным численным методам и часто используемым алгоритмам решения некоторых задач программирования. Обычно семинары проводятся раз в две недели по очередным элементам синтаксиса, после чего выдается очередная задача практикума.

Полученные задания студенты выполняют на практических занятиях в компьютерных классах: используя установленную там среду программирования студенту нужно самостоятельно написать и отладить программу, решающую выданную ему задачу, после чего сдать эту программу преподавателю. Преподаватель во время сдачи задания может задать студенту вопросы по любой части представленной программы, а также попросить при нем выполнить ее небольшую модификацию, чтобы убедиться в том, что студент полностью понимает сдаваемую им программу и способен самостоятельно ее развивать.

В течение семестра студенты должны выполнить 6 заданий, на каждое из которых отводится по 4 часа практикума. В данном пособии не предполагается строгое соответствие между описанными задачами и содержанию заданий. Некоторые главы содержат более одной задачи. Соответствующее задание может тоже содержать несколько задач. В этом случае рекомендуется помещать их решение внутри одной функции *main*() выделив комментариями, где какая задача расположена.

В середине семестра рекомендуется провести коллоквиум в форме тотального опроса всех студентов группы по пройденному материалу и выставить оценку промежуточной аттестации. В конце семестра проводится зачет, который в обязательном порядке включает как ответ на вопросы по теоретическому минимуму языка программирования, так и решение практической задачи, подобной тем задачам, которые студенты решали в течение семестра.

Перед началом работы в практикуме рекомендуется провести устный опрос студентов с целью выяснить какой опыт программирования у них был ранее, на каких языках программирования им приходилось писать программы? И приходилось ли вообще? До сих пор иногда встречаются студенты с очень ограниченными и даже отсутствующими навыками работы даже с самим персональным компьютером, таким студентам нужно уделять больше внимания в практикуме.

Важное замечание: в компьютерных классах практикума может быть установлена более старая версия среды разработки, по сравнению с современными версиями, актуальными в настоящий момент. Однако, для целей учебного процесса первого курса возможностей версии среды разработки, установленной в практикуме, более чем достаточно. При этом мы не приветствуем использование студентами в практикуме первого курса своих ноутбуков с какими-либо другими версиями среды разработки. Причина этого заключается в том, что все студенты должны быть в равном положении (вне зависимости от наличия или отсутствия у них ноутбуков), в том числе они будут должны на зачете продемонстрировать свои знания и навыки именно в той среде, которая установлена в практикуме, а для этого они должны будут получить устойчивые навыки работы в среде практикума в течение семестра. На следующих курсах это ограничение может быть снято.

Часто студенты задают вопрос о разработке ими программ дома. Тут есть некоторая сложность: требования безопасности запрещают студентам подключать какие-либо устройства (включая flash-диски) к компьютерам практикума. Преподаватель по своему усмотрению со своего компьютера может помочь студенту скопировать его исходные тексты из класса на flash-диск студента, но не в обратном направлении! Решением является использование студентом электронной почты: каждый студент получает свой персональный e-mail, доступ к которому возможен через браузер как с компьютера в практикуме, так и с домашнего компьютера студента. Однако, преподаватель должен следить за тем, чтобы студенты понимали сдаваемые ими программы: довольно часто студенты пользуются посторонней помощью при такой удаленной работе, что может привести к большим проблемам на зачете, где такой помощи не будет. И подчеркнем, что работа программы у студента дома вовсе не означает её беспроблемную компиляцию и запуск в классе практикума из-за различий в среде разработке, версии компилятора и поддерживаемых им стандартов языка программирования.

Глава 1. Первая программа.

Особенности языка Си

Программа на языке Си является **структурно ориентированной** и состоит из набора функций и переменных. Функции содержат набор инструкций, описывающих вычисления, которые необходимо выполнить. А переменные хранят значения, которые используются в процессе вычислений.

Структурный подход предполагает алгоритмическое **разбиение задач на подзадачи** и следование следующим правилам:

- Любая программа может быть представлена использованием трёх базовых управляющих конструкций (последовательные операции, ветвление, цикл).
- Повторяющиеся фрагменты оформляются в виде отдельных функций.
- Каждая логически законченная структура оформляется в блок.
- Базовые конструкции могут быть вложены друг в друга.
- Отказ от оператора безусловного перехода *goto*.

Разработчик может определять любые имена для своих функций. Но в программе должна присутствовать функция, с которой начинается исполнение программы. В консольном приложении для нее зарезервировано имя *main*.

Следующие за именем функции круглые скобки предназначены для задания списка параметров (или аргументов), которые передаются в функцию при обращении к ней. Если внутри скобок ничего не указано, то в функцию не передается никаких аргументов. Функция *main* может содержать два параметра, позволяющее передать в функцию текст, который был напечатан в командной строке при запуске программы. Её заголовок тогда выгладит так:

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

argc определяет количество переданных параметров (если передано только имя программы, то *argc* равно 1), *argv* – массив строковых параметров.

Перед именем функции указывается тип возвращаемого значения. Функция *main()* должна возвращать целочисленное значение (*int*). После завершения программы это значение можно будет увидеть в строке сообщения среды разработки. Для этого в коде перед завершением функции *main()* нужно добавить строку:

```
return 0;
```

Число ноль принято интерпретировать как успешное завершение программы.

Каждый оператор в языке Си заканчивается символом «точка с запятой». В качестве оператора может выступать вызов функции или осуществление некоторых операций. Для написания комментариев используются следующие символы:

`/*` - начало комментария,

`*/` - конец комментария.

При этом все что будет заключено между этими символами будет являться комментариями, то есть не будет восприниматься компилятором как инструкция или набор инструкций. Такое ограничение области комментариев удобно использовать при комментировании блока программы, состоящего из нескольких строк. При необходимости комментария одной строки удобно использовать символы //, при этом комментарием будет являться все, что расположено между символами // и концом строки (т.е. все, что расположено справа от данных символов).

Функции для выполнения своей работы могут вызывать другие функции, которые либо пишутся самим программистом, либо берутся готовыми из имеющихся библиотек. Для подключения объявлений функций из системных библиотек используется команда препроцессора *#include*, например

```
#include <stdio.h>
```

подключает объявления функций стандартной библиотеки ввода-вывода.

Стоит также обратить внимание на возможность использования русских шрифтов при вводе-выводе. Дело в том, что кодировка консольного приложения настроена на использование старой 866 кодовой таблицы символов, которая не совпадает с таблицей windows. Проблему можно решить несколькими способами.

Наиболее простой способ – вызвать функцию установки русской локали:

```
setlocale(LC_ALL, "rus");
```

Однако, у этого способа есть два нюанса. Функция влияет только на вывод, а ввод все равно останется в старой кодировке. И, кроме того, русская локаль в языке Си предусматривает для вещественных чисел указание десятичной запятой, а не десятичной точки.

Другой способ – это использовать консольную команду, устанавливающую для ввода и вывода кодовую страницу windows:

```
system("chcp 1251 > NUL");
```

В этом случае ввод и вывод русских букв будет корректным. Единственное, при первом открытии консоли нужно в ее системном меню выбрать шрифт «*Lucida Console*».

Примеры простых программ на языке Си

Рассмотрим несколько простых примеров программ. Если какие-то из команд будут непонятны, то это сейчас неважно. Они будут рассмотрены в следующих темах.

Пример 1. Сравнение двух чисел.

Программа берет два числа либо из консольной командной строки, если они там есть, либо запрашивает их у пользователя программы и далее определяет, какое из них больше.

```
#include <stdio.h>  // подключение заголовочного файла ввода-вывода
#include <stdlib.h> // подключение заголовочного файла стандартной библиотеки
#include <locale.h> // подключение заголовочного файла локализации

// начало функции main являющейся точкой входа в программу
// параметры функции: argc - количество параметров, переданных в командной строке
// argv - адреса переданных слов
int main(int argc, char* argv[])
{
	int a, b;  // объявление двух переменных типа integer
	setlocale(LC_ALL, "rus"); // для вывода русских строк
	if(argc<3) // проверяем содержимое командной строки
	{
		// в командной строке недостаточно данных
		printf("Введите число 1: "); // запрос на ввод 1 числа
		scanf("%d", &a); //считывание числа в переменную a
		printf("Введите число 2: "); // запрос на ввод 2 числа
		scanf("%d", &b); //считывание числа в переменную b
		getchar();// считывание введенного, но уже ненужного символа enter
	} else
	{
		// числа можно забрать из командной строки
		// читаем в a число из второго параметра командной строки
		a = atoi(argv[1]);
		// читаем в b число из третьего параметра командной строки
		b = atoi(argv[2]);
	}
	if (a > b) // условный оператор, проверяющий верно ли, что a>b
	{
		printf("Первое число (%d) больше второго (%d)\n",a,b);
	}else
	{
		if (a == b) // условный оператор,проверяющий верно ли,что a равно b
		{
			printf("Числа равны(%d)=(%d)\n",a,b);
		}
		else
		{
			printf("Второе число (%d) больше первого (%d)\n",b,a);
		}
	}
	getchar();// ожидаем ввода символа, чтобы терминал не закрылся
	return 0;
} // конец функции main
```

Пример 2. Использование оператора множественного выбора.

Программа выполняет расчет периметра квадрата, площади квадрата или объема куба используя оператор *switch*.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[])
{
	char b; // объявление символьной переменной
	double a,res; // объявление действительных переменных двойной точности
```

```
	system("chcp 1251 > NUL"); // настройка на ввод и вывод русского текста
	if(argc<2) // проверка наличия числа в командной строке
	{
		// в командной строке нет данных
		printf("Введите длину стороны квадрата: ");
		scanf(" %lf", &a); // ввод длины стороны с клавиатуры
		getchar();// считывание ненужного символа enter
	} else
		a = atof(argv[1]); // ввод числа из командной строки

	printf("Введите параметр расчета: S-площадь, V - объём, P - периметр:");
	scanf(" %c", &b); // вид расчета всегда вводим с клавиатуры
	getchar();// считывание введенного символа enter
	switch(b)
	{
	case 'S':
		res=a*a; // вычисление площади
		printf("Площадь - %lf\n", res);
		break;
	case 'V':
		res=a*a*a; // вычисление объема
		printf("Объем - %lf\n", res);
		break;
	case 'P':
		res=4.*a; // вычисление периметра
		printf("Периметр  - %lf\n", res);
		break;
	default:
		printf("Ошибка\n");
		break;
	}
	getchar();// ожидаем ввода символа, чтобы терминал не закрылся
	return 0;
}
```

Пример 3. Вычисление логических побитовых операций

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>

int main()
{
	unsigned int a, b, c, d, e, f, g, h;
	//объявление беззнаковых переменных типа integer
	setlocale(LC_ALL, "rus"); // переходим в консоли на русский язык
	a = 15;
	b = 136;
	c = ~a;   // not a
	d = a<<1; // сдвиг a на 1 разряд влево
	e = a>>1; // сдвиг a на 1 разряд вправо
	f = a&b;  // a and b
	g = a^b;  // a xor b
	h = a | b;// a or b
	printf("Операция ~: %d\n", c);
	printf("Операция <<: %d\n", d);
	printf("Операция >>: %d\n", e);
	printf("Операция &: %d\n", f);
	printf("Операция ^: %d\n", g);
	printf("Операция |: %d\n", h);
	getchar();
	return 0;
}
```

Пример 4. Расчет суммы чисел от 1 до 100 с использованием цикла.

Программа использует цикл *do{..}while (условие окончания цикла).* Цикл повторяется до тех пор, пока

условие не станет ложным.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
int main()
{
setlocale(LC_ALL, "rus"); // переходим в консоли на русский язык
	int i,sum; //объявление переменных типа int
	i=1;		// начальное значение i равно 1
	sum=0;		// начальное значение sum рвно 0
	do //повторять
	{
		sum += i; // увеличить sum на i
		i++;		// увеличить  i на 1
	}while (i<101); // пока значение i не станет равным 101
	printf("Сумма чисел от 1 до 100 = %d", sum);//вывод суммы
	getchar();// ожидаем ввода символа, чтобы терминал не закрылся
	return 0;
}
```

Пример 5. Поиск максимума во введенной последовательности целых чисел.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
int main()
{
	int a,amax;
	setlocale(LC_ALL, "rus"); // переходим в консоли на русский язык
	printf("Введите первое число последовательности:\n");
	scanf(" %d",&a);
// начальное значение максимума берем равным первому введенному числу:
	amax = a;
	printf("Вводите числа последовательности(0 - окончание ввода):\n");
//повторять, пока нет ошибок ввода и введенное число не равно нулю:
	while (scanf(" %d",&a)==1 && a!=0)
	{
		if (amax < a)// сравниваем очередной введенный элемент и макс.
			amax = a; // берем больший из них
	}
	getchar(); // убираем символ ENTER
	printf("Максимум = %d\n", amax);
	getchar();// ожидаем ввода символа, чтобы терминал не закрылся
	return 0;
}
```

Варианты задач для решения.

Для выполнения задач необходимо создать консольный проект и добавить в него один новый *task1.cpp* файл. Необходимые для ввода данные нужно читать из командной строки, если их окажется недостаточно, то запрашивать у пользователя (аналогично тому, как сделано в первом примере). При отладке программы из Visual Studio данные командной строки нужно записать в командные аргументы проекта (команда меню Проект–>свойства–>свойства конфигурации –> отладка). Подробно эти действия описаны в главе 2 пособия «Язык программирования Си».

1. Вариант Дано целое положительное число. Написать программу, проверяющую истинность высказывания: «Данное число является четным двузначным».
2. Вариант Даны три целых числа: A, B, C. Написать программу, проверяющую истинность высказывания: «Каждое из чисел A, B, C положительное».
3. Вариант Написать программу, спрашивающую у пользователя несколько чисел (больше трёх) и выводящую их среднее значение и дисперсию (среднее квадрата отклонения от среднего значения).

4. Вариант
Написать программу, проверяющую, что, либо A четно, либо B четно (но не оба одновременно).

5. Вариант
Написать программу, спрашивающую у пользователя три числа и выводящую их в порядке возрастания.

6. Вариант
Даны три целых числа: A, B, C. Написать программу, проверяющую истинность высказывания: «Справедливо двойное неравенство A < B < C».

7. Вариант
Даны два целых числа: A и B. Написать программу, проверяющую истинность высказывания: «Числа A и B имеют одинаковую четность».

8. Вариант
Написать программу, проверяющую, что среди чисел A, B, C и D есть как минимум 2 нечетных.

9. Вариант
Используя оператор switch, написать программу вывода прописью значения введенной переменной, если ее значение в диапазоне от 1 до 5. В остальных случаях значение вывести цифрами.

10. Вариант
Даны два целых числа A и B (A < B). Написать программу, выводящую в порядке убывания все целые числа, расположенные между A и B (не включая числа A и B).

11. Вариант
Даны два целых числа A и B (A < B). Написать программу, находящую произведение всех целых чисел от A до B включительно.

12. Вариант
Дано целое число N (> 1). Написать программу, находящую наименьшее целое число K, при котором выполняется неравенство 3K > N.

13. Вариант
Даны координаты трех точек на плоскости. Написать программу, вычисляющую значение линейной интерполяции или экстраполяции для любого введённого x по двум уравнениям прямых, проведенных через соседние точки.

14. Вариант
Написать программу, вычисляющую площадь треугольника по трем сторонам. Если треугольник с такими сторонами не существует, то вывести сообщение об этом.

15. Вариант
Написать программу, проверяющую является ли прямоугольным треугольник с заданными сторонами. Если треугольник с такими сторонами не существует, то вывести сообщение об этом.

16. Вариант
Написать программу, вычисляющую площадь треугольника по заданным координатам его вершин.

17. Вариант
Написать программу, поверяющую является ли равносторонним треугольник с заданными координатами его вершин.

18. Вариант
Написать программу, подсчитывающую сумму всех нечетных чисел от 1 до указанного N.

19. Вариант
Написать программу подсчитывающую количество десятичных цифр введенного натурального числа.

20. Вариант
Известно, что сумма N первых нечетных чисел равна квадрату числа N, например, $1 + 3 + 5 = 3^2$, $1 + 3 + 5 + 7 = 4^2$ и т.д. Написать программу, выводящую таблицу всех натуральных чисел от 1 до K и их квадратов, вычисленных с помощью указанного соотношения.

21. Вариант
Написать программу возведения числа в степени 2,3,4 или 5 используя оператор switch.

22. Вариант Написать программу, вычисляющую значение факториала $N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$.
23. Вариант Написать программу расчета суммы чисел в диапазоне от n до N. Числа n и N должны вводиться с клавиатуры.
24. Вариант Написать программу вывода всех чисел в диапазоне от 0 до указанного N кратных 3 или 5.
25. Вариант Написать программу расчета суммы введенной последовательности чисел. Первым вводимым параметром задаётся количество чисел. Если числа заданы в командной строке, то их количество равно *argc* – 1.
26. Вариант Написать программу проверки того, что элементы введенной последовательности чисел не убывают. Первым вводимым параметром задаётся количество чисел. Если числа заданы в командной строке, то их количество равно *argc* – 1.
27. Вариант Написать программу, которая выводит попарные суммы вводимой последовательности чисел. Первым вводимым параметром задаётся количество чисел. Если числа заданы в командной строке, то их количество равно *argc* – 1.
28. Вариант Написать программу, которая вычисляет разность между средним и минимальным значениями во вводимой последовательности. Первым вводимым параметром задаётся количество чисел. Если числа заданы в командной строке, то их количество равно *argc*–1.
29. Вариант Написать программу, которая во вводимой последовательности определяет количество чисел равных заданному значению (задать его в виде константы). Первым вводимым параметром задаётся количество чисел. Если числа заданы в командной строке, то их количество равно *argc*–1.
30. Вариант Написать программу суммирования всех десятичных чисел в диапазоне от 1 до указанного N, у которых последняя цифра больше 5.

Глава 2. Основы синтаксиса языка Си.

В данной главе не требуется (но допустимо) использование функций, которые будут изучаться позднее.

Методы решения уравнений

Если задано уравнение `f(x) = 0`, то с помощью компьютера его можно решить приближенно, практически с любой нужной точностью. Для этого можно применить один из описываемых ниже методов: метод дихотомии, метод хорд, метод касательных или метод итераций.

Подробно методы решения уравнений излагаются на лекциях (см. В.А. Антонюк, А.П. Иванов. «Программирование и информатика. Краткий конспект лекций.» – Москва, физический ф-т МГУ, 2015, (https://cmp.phys.msu.ru/sites/default/files/Informatics-2015.pdf, глава 2).

Методы касательных и итераций позволяют получить решение гораздо быстрее (за меньшее число итераций), чем методы дихотомии и хорд, однако, применимы они не к любым уравнениям: для того, чтобы их применить, нужно, чтобы выполнялись некоторые условия, накладываемые на производные функции, которая задает левую часть уравнения. Если такие условия не выполняются – то придется пользоваться менее эффективными, но гарантированно сходящимися методами: методом дихотомии или методом хорд.

Точностью решения будем называть найденную длину интервала по оси OX, гарантированно накрывающего истинное решение уравнения. Однако, такую величину можно явно получить только для первого метода решения – метода деления интервала пополам. Для остальных методов в качестве приближенной оценки точности будем использовать расстояние между двумя последовательными приближениями решения $|x_{i-1} - x_i|$.

Такой подход оправдан, так как описываемые методы обеспечивают монотонную сходимость приближений решения к истинному корню, а следовательно, расстояние между такими последовательными приближениями должно монотонно убывать и рано или поздно окажется меньше требуемой точности решения.

Невязкой решения называется модуль результата подстановки найденного решения в левую часть уравнения (для метода итераций это модуль разности такой подстановки и правой части уравнения). Другими словами, невязку можно назвать «точностью по оси OY». Невязка легко вычисляется для любого из приведенных методов решения уравнения.

Отметим, что в ходе численного решения уравнений в данном задании нужно добиться одновременного удовлетворения заданных условий и на точность решения и на его невязку.

Метод дихотомии

Метод дихотомии, он же метод деления отрезка пополам, он же метод вилки заключается в следующем:

1) Получаем от пользователя начальный интервал `[a, b]`, требуемую точность δ и невязку ε.
2) Проверяем, что на концах этого интервала функция левой части уравнения имеет разные знаки: `f(a) * f(b) < 0` ?
3) Если нет – то решения на заданном интервале нет, либо оно не единственное, либо вообще функция имеет на нём разрыв. Печатаем диагностику и завершаем программу. Пользователь вашей программы должен попробовать подобрать начальный интервал получше.
4) Если знаки на концах интервала разные, то находим середину отрезка:
$$c = \frac{a+b}{2}$$
5) Проверяем, на каком из двух интервалов `[a, c]` или `[c, b]` функция меняет знак: `f(a) * f(c) < 0` или `f(c) * f(b) < 0` ?
6) Если функция меняет знак на первом интервале, то присваиваем переменной `b` значение `c`: `b = c`.
7) Иначе: `a = c`. Однако и для этого интервала нужно убедиться, что функция меняет на нём знак, так как функция может иметь разрывы и тогда решение построить не получится.
8) Повторяем п.п. 4-7 до того момента, когда интервал окажется меньше заданной точности δ и одновременно невязка уравнения окажется меньше заданного ε.

Точность данного метода определяется шириной интервала `|b-a|` на последней итерации, который гарантированно накрывает искомый корень.

Невязкой уравнения называется модуль подстановки текущего приближения корня `c` в уравнение: `|f(c)|`.

Таким образом, условие, при котором надо прекратить итерации, записывается так:

`|b-a| < δ && |f(c)| < ε`

Однако, обычно для построения данного решения используется цикл `while()` (продолжать, пока условие истинно), то есть, для записи условия такого цикла нужно построить логическое отрицание

приведенного выше выражения – итерации нужно продолжать до тех пор, пока истинно:

`|b-a| ≥ δ || |f(c)| ≥ ε`

Такой подход следует использовать во всех приведенных методах решения уравнений, так как во всех заданиях требуется одновременное удовлетворение условий, заданных и для точности и для невязки, только точность в последующих методах определяется как расстояние между двумя последовательными приближениями корня, а не как в данном методе.

```
#include <stdio.h>
#include <math.h>
.............
    // Пусть решается уравнение: atan(x) - 1 = 0
    // (atan() – стандартная функция арктангенса)
    // fabs() – стандартная функция, вычисляющая модуль аргумента
    double c = a;
    while ( fabs(b-a) >= delta || fabs(c) >= epsilon )
    {
          c = (a + b) / 2; // находим середину интервала
          if ( (atan(a)-1) * (atan(c)-1) < 0 )
          {
                b = c; // если функция меняет знак на первом интервале
          }else if ( (atan(c)-1) * (atan(b)-1) < 0 )
          {
                a = c; // если функция меняет знак на втором интервале
          }else
          {
                /* функция не меняет знак ни на одном интервале, она имеет разрывы
                или множественные корни на каждом из них, решение построить
                невозможно */
                printf("Error: cannot solve equation on [%lf, %lf]!", a, b);
                return 1; // завершаем функцию main() с кодом ошибки
          }
    }
.............
```

Метод хорд

Метод хорд очень похож на метод дихотомии, но часто достигает требуемого результата немного быстрее. Основное отличие заключается в способе разделения отрезка на две части:

1) Получаем от пользователя начальный интервал `[a, b]`, требуемую точность δ и невязку ε.
2) Проверяем, что на концах этого интервала функция левой части уравнения имеет разные знаки: `f(a) * f(b) < 0`?
3) Если нет – то решения на заданном интервале нет, либо оно не единственное. Печатаем диагностику и завершаем программу. Пользователь вашей программы должен попробовать подобрать начальный интервал получше.
4) Если знаки на концах интервала разные, то находим точку пересечения прямой, соединяющей точки `(a, f(a))` и `(b, f(b))`:

$$c = a + \left|\frac{f(a)}{f(b) - f(a)}\right| \cdot (b - a)$$

5) Проверяем, на каком из двух интервалов `[a, c]` или `[c, b]` функция меняет знак: `f(a) * f(c) < 0` или `f(c) * f(b) < 0` ?
6) Если функция меняет знак на первом интервале, то присваиваем переменной `b` значение `c`: `b = c`.
7) Иначе: `a = c`. Однако и для этого интервала нужно убедиться, что функция меняет на нём знак, так как функция может иметь разрывы и тогда решение построить не получится.
8) Повторяем п.п. 4-7 до того момента, когда расстояние между двумя приближениями корня окажется меньше заданной точности δ и одновременно невязка уравнения окажется меньше заданного ε.

Для многих уравнений ширина интервала, на котором ведется поиск корня будет стремиться не к нулю, а к константе, поэтому тут точность решения будем оценивать, как расстояние между двумя последовательными приближенными решениями. Эта величина будет стремиться к нулю при увеличении количества итераций.

Метод касательных

Метод касательных называется также методом Ньютона, он заключается в следующем:

1) Получаем от пользователя начальное приближение x_0 (номер итерации `i=0`), требуемую точность δ и невязку ε.

2) Проверяем условие сходимости $f(x_i) \cdot f''(x_i) > 0$ (то есть, проверяем, что функция постоянно выпуклая или вогнутая).
3) Если условие сходимости нарушено – печатаем диагностику и завершаем программу, метод не применим к данному уравнению на данной и последующих итерациях. Можно для диагностики выдать номер итерации и достигнутые значения корня, точности и невязки.
4) Если условие сходимости выполняется, то любое последующее уточнение значения корня получаем по итерационной формуле:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Эта формула получена как точка пересечения касательной в точке x_i к графику функции `f(x)` с осью `OX`.

5) Повторяем п.п. 2-4 до того момента, когда ширина интервала $[x_{i+1}, x_i]$ окажется меньше заданной точности δ и одновременно невязка уравнения окажется меньше заданного ε.

Важно: условие сходимости обязательно нужно проверять на каждой итерации цикла, а не только перед первой итерацией. Действительно, выпуклая функция может перестать быть выпуклой не сразу, а спустя сколько-то шагов движения к корню.

Метод итераций

Метод итераций применим и в тех случаях, когда условие сходимости метода касательных не выполняется, но на функцию левой части уравнения может быть наложено другое условие.

Перепишем уравнение в следующем виде:

$$F(x) = x$$

Тогда, если выполняется условие сжимающего отображения:

$$|F'(x)| < 1$$

То уравнение можно решать так:

1) Получаем от пользователя начальное приближение x_0 (номер итерации `i=0`), требуемую точность δ (невязка для данного метода не нужна, так как по построению она совпадает с оценкой точности).
2) Проверяем условие сходимости $|F'(x_i)| < 1$.
3) Если условие сходимости нарушено – печатаем диагностику и завершаем программу, метод не применим к данному уравнению на данной и последующих итерациях. Можно для диагностики выдать номер итерации и достигнутые значения корня, точности и невязки.
4) Если условие сходимости выполняется, то любое последующее уточнение значения корня получаем по итерационной формуле:

$$x_{i+1} = F(x_i)$$

5) Повторяем п.п. 2-4 до того момента, когда ширина интервала $[x_{i+1}, x_i]$ окажется меньше заданной точности δ.

Важно: условие сходимости обязательно нужно проверять на каждой итерации цикла, а не только перед первой итерацией. И поясним, почему для данного метода не нужна невязка – оценка точности решения уравнения одновременно является и его невязкой:

$|x_{i+1} - x_i| = |F(x_i) - x_i|$

Варианты задач для решения

Одним из описанных методов решить заданное уравнение.

Программа должна получить начальные значения переменной (начальный интервал для методов дихотомии и хорд, единственное начальное значение для методов касательных и итераций, требуемую точность и требуемую невязку, которые должны быть удовлетворены одновременно).

После этого программа должна произвести вычисление и напечатать:

1) *Исходное уравнение, исходный интервал и заданную точность;*
2) *Найденный корень;*
3) *Достигнутую точность: ширину последнего шага итерации (или расстояние между двумя последними приближениями корня);*
4) *Невязку: результат подстановки найденного неточного решения в левую часть уравнения;*
5) *Количество проделанных до достижения требуемой точности итераций цикла.*

Требуется, чтобы программа была устойчива к задаваемому пользователем интервалу, если корней вообще нет или корень не единственный или не выполняются условия сходимости для методов касательных и итераций нужно напечатать соответствующую диагностику. Не забывать также выдавать ошибки, если начальный интервал выходит за область допустимых значений функции уравнения.

Для методов касательных и итераций производные заданной функции найти аналитически.

Для предварительной оценки количества и расположения корней уравнения рекомендуется построить

график функции уравнения: для этого можно воспользоваться любой программой электронных таблиц, для которой можно отдельно из своей программы распечатать значения нужной функции в достаточно широкой окрестности нуля, вычисленные с равномерным шагом по оси OX.

1. Вариант

Решить методом **дихотомии** уравнение:`ln(x) - 1/x = 0`

Функция натурального логарифма в Си: `log(x)`.

2. Вариант

Решить методом **хорд** уравнение: `exp(x) + x = 0`

Решить методом **касательных** уравнение:`1/x - 2*x = 0`

При анализе введенных данных учесть разрывность функции – если начальный интервал включает разрыв, то можно сразу выдавать ошибку.

Решить методом **дихотомии** уравнение: `ln(x) + 1 = 0`

Функция натурального логарифма в библиотеке языка Си: `log(x)`.

3. Вариант

Решить методом **хорд** уравнение: `arctg(x) - 1/2 = 0`

Функция арктангенса в библиотеке языка Си: `atan(x)`.

4. Вариант

Решить методом **касательных** уравнение: `exp(x) - 1/2 = 0`

5. Вариант

Решить методом **итераций** уравнение: `3 - ln(x-1) = x`

Обратить внимание, что условие сходимости выполняется не для любого интервала, входящего в область допустимых значений, необходимо это обнаруживать и выдавать соответствующее сообщение об ошибке. Корней у уравнения несколько и не любой из них можно найти предлагаемым методом. Нужно самостоятельно оценить, с какого начального приближения можно начинать искать возможные корни. Функция натурального логарифма в библиотеке языка Си: `log(x)`.

6. Вариант

Решить методом **касательных** уравнение: `exp(x) + ln(x) = 0`

Обратить внимание: условие сходимости нарушается для значений x, меньших корня. Можно уменьшить левую часть уравнения на константу 10 и оценить, как это влияет на допустимые начальные значения.

Функция натурального логарифма в библиотеке языка Си: `log(x)`.

7. Вариант

Решить методом **дихотомии** уравнение: `tg(2*x) – 1 - x = 0`

Обратить внимание на места разрывов функции, контролировать введенный начальный интервал.

Функция тангенса в библиотеке языка Си: `tan(x)`.

8. Вариант

Решить методом **хорд** уравнение: `2*exp(-x) - x = 0`

9. Вариант

Решить методом **касательных** уравнение: `exp(-3*x^2) - x - 1= 0`

Обратить внимание: у функции три близких корня и условие сходимости выполняется в довольно узкой области каждого из них. Нужно самостоятельно оценить, с какого начального приближения можно начинать искать каждый из трех корней.

10. Вариант

Решить методом **итераций** уравнение:

`tg(x+0.1)/3 = x`

Обратить внимание: функция периодическая и разрывная и на каждом периоде есть три близко лежащих корня, при этом окрестность только одного из них удовлетворяет условию сходимости. Нужно самостоятельно оценить, с какого начального приближения можно начинать искать этот корень. Для этого рекомендуется отдельно построить графики функции и ее производной.

Функция тангенса в библиотеке языка Си: `tan(x)`.

11. Вариант
Решить методом **дихотомии** уравнение: `exp(-3*x) + 1 - x = 0`

12. Вариант
Решить методом **хорд** уравнение: $\exp(-x^2) - x = 0$

13. Вариант
Решить методом **касательных** уравнение: $\exp(-x^2) - x^2 = 0$
Обратить внимание: функция имеет два близко лежащих корня и условие сходимости выполняется в достаточно узкой области каждого из них. Нужно самостоятельно оценить, с какого начального приближения можно начинать искать каждый из двух корней.

14. Вариант
Решить методом **касательных** уравнение: `2*exp(-3*x) + 1 - x = 0`
Обратить внимание: условие сходимости выполняется только слева от корня, проверить это.

15. Вариант
Решить методом **дихотомии** уравнение: `3*exp(-3*x) - x = 0`

16. Вариант
Решить методом **хорд** уравнение: $\exp(-x^3) - 1 - x^3 = 0$

17. Вариант
Решить методом **итераций** уравнение: `0.4*cos(2*x) = x`

18. Вариант
Решить методом **хорд** уравнение: `tg(x/2)/10 - 1 - x = 0`
Функция тангенса в библиотеке языка Си: `tan(x)`.

19. Вариант
Решить методом **касательных** уравнение: `3*cos(x) - x = 0`
Обратить внимание: уравнение имеет два близко лежащих корня и третий корень – чуть подальше. При этом условия сходимости для первых двух корней выполняются в довольно узкой их области, а для третьего корня условие сходимости не выполняется. Нужно самостоятельно оценить начальное приближение для первых двух корней и проверить, что третий корень найти методом касательных нельзя. .

20. Вариант
Решить методом **итераций** уравнение: `exp(-x) = x`
Обратить внимание: справа от корня условие сходимости выполняется всегда, а вот слева от корня – только на небольшом расстоянии от него. Самостоятельно определить, с какого минимального начального приближения (слева от корня) имеет смысл начинать его поиск?

21. Вариант
Решить методом **дихотомии** уравнение: `exp(-3*x) - x = 0`

22. Вариант
Решить методом **итераций** уравнение: $\exp(-x^2) + 2 = x$
Проверить, что условие сходимости выполняется всюду, поэтому начинать можно с любого начального приближения.

23. Вариант
Решить методом **хорд** уравнение: `1/x - x/2 = 0`
Обратить внимание на разрывность функции, выдавать ошибку, если начальный интервал включает в себя разрыв.

24. Вариант
Решить методом **дихотомии** уравнение: $(x^2 - 1)/(x^2 + 1) = 0$

25. Вариант

Решить методом **хорд** уравнение: $(\exp(-2x) - 2)/(\exp(-x) + 1) = 0$

26. Вариант

Решить методом **касательных** уравнение: $3 \cdot \ln(x^2 + 1) - 1 = 0$

Обратить внимание: уравнение имеет два близко лежащих корня и условие сходимости выполняется в довольно узких интервалах вокруг каждого из них. Самостоятельно определить, с какого начального приближения нужно начинать искать каждый из корней.

Функция натурального логарифма в Си: `log(x)`.

27. Вариант

Решить методом **касательных** уравнение: $\cos(x) + \cos(x^2) - 1.55 = x$

Обратить внимание: уравнение имеет несколько близко лежащих корней, однако, условие сходимости метода выполняется в довольно узкой окрестности нескольких, но не всех корней. Нужно самостоятельно подобрать начальные приближения для нахождения каждого из корней, в окрестностях которых выполняются условия сходимости данного метода.

Функция натурального логарифма в Си: `log(x)`.

28. Вариант

Решить методом **дихотомии** уравнение: $3 \cdot \ln(x^2 + 1) - \ln(x^3 + 1) - 1 = 0$

Уравнение имеет два корня, самостоятельно подобрать начальные интервалы для нахождения каждого из них.

29. Вариант

Решить методом **дихотомии** уравнение: $3\exp(-3x) - x = 0$

30. Вариант

Решить методом **хорд** уравнение: $\exp(-2x) - x = 0$

Глава 3. Массивы.

Методы сортировки и поиска

Сортировка — это преобразование массива неупорядоченных элементов в массив упорядоченных по какому-либо критерию элементов. В простейшем случае элементами массива могут выступать числа, а сортировка производится для расположения чисел по возрастанию.

Подробно основные методы сортировки и поиска излагаются на лекциях (см. В.А. Антонюк, А.П. Иванов. «Программирование и информатика. Краткий конспект лекций.» – Москва, физический ф-т МГУ, 2015, (https://cmp.phys.msu.ru/sites/default/files/Informatics-2015.pdf, главы 4-5).

«Пузырьковая» сортировка

Одна из простейших для понимания сортировок. Получила свое название из-за того, что на каждой итерации наибольший элемент оказывается на своей позиции в массиве, «всплывая», как пузырек.

2. Последовательно для каждой очередной пары элементов массива (0; 1), (1; 2), (2; 3), …, (N-2; N-1) производится сравнение. Если в паре элементы расположены не по возрастанию, то производится обмен значений элементов пары. В процессе выполнения первого шага будет «зацеплен» максимальный элемент массива, который далее просто «всплывет» до последней позиции массива.
3. Поэтому далее повторяется первый шаг, но при этом каждый раз уменьшается количество рассматриваемых пар элементов на 1, так как крайний правый элемент уже оказался на своей позиции и его не надо более ни с кем сравнивать. Соответственно, второй повтор первого шага должен будет дойти до N-2 элемента массива, третий – до N-3 и так далее.
4. Алгоритм завершается, когда он выполнит единственное сравнение для первой пары элементов массива, то есть, длина неотсортированной части массива сократится до нуля.

Немного быстрее будет работать модифицированная версия этого алгоритма, когда направления «прогона» пузырька меняются после каждой итерации: на первой итерации «всплывает» максимальный элемент массива, на второй – «тонет» минимальный элемент, н третьей – опять всплывает максимальный из оставшихся и так далее. Таким образом неотсортированная часть будет уменьшаться с обоих концов. Такая модификация называется шейкерной сортировкой.

Сортировка выбором

На каждой итерации выбирается наименьший элемент массива.

1. Производится поиск наименьшего элемента массива. Как только этот элемент найден, производится обмен этого значения с первым неотсортированным элементом. Тем самым, массив оказывается разбит на две части: отсортированную (пока из одного минимального элемента) и неотсортированную (все остальные элементы, начиная со второго).
2. Повторяется первый шаг для оставшихся элементов неотсортированной части: начиная с элемента 2, на следующей итерации – с элемента 3 и так далее.
3. Алгоритм завершается, как только в неотсортированной части останется только один элемент.

Сортировка вставкой

Массив делится на две части: уже отсортированную (вначале в ней только один первый элемент массива) и еще не отсортированную (все остальные элементы массива).

На каждой итерации очередной элемент неотсортированной части массива вставляется на нужную позицию в отсортированной части.

1. Взять очередной первый элемент из неотсортированной части массива и найти место для его вставки в отсортированной части (последовательным перебором или методом деления отсортированной части пополам): нужно найти первый элемент отсортированной части, который будет меньше выбранного, тогда позицией для вставки будет позиция следующего за ним элемента.
2. Начиная с найденной позиции весь хвост отсортированной части массива сдвинуть на одну позицию вправо (и тем самым затереть выбранный в п.1 элемент).
3. Сохранить выбранный элемент в найденную позицию (затереть то, что там было ранее). Тем самым, отсортированная часть увеличилась на один элемент, а неотсортированная – уменьшилась на один элемент.
4. Повторить шаги 1-3 для оставшихся элементов входного массива.
5. Алгоритм завершается, когда в неотсортированной части массива не останется ни одного элемента.

Метод ускоренной сортировки Шелла

В качестве примера программного кода алгоритмов сортировки приведем алгоритм сортировки Шелла, гораздо более быстрый по сравнению с изложенными выше методами:

```
#include <stdlib.h>
#define N 100
….
```

```
int A[N];
int i,j;
for (i = 0; i < N; ++i)
      A[i] = rand(); // Подготовим данные: заполним массив случайными числами
// Сортировка Шелла
int gap;
// Будем сортировать подмассивы с элементами на расстоянии gap друг от друга
for (gap = N/2; gap > 0; gap /= 2) // gap стремится к 1
{
      for (i = gap; i < N; ++i)// Переберём все такие подмассивы
      {
// внутренний цикл – это одна итерация сортировки вставками одного подмассива
            for (j = i - gap; j >= 0 && A[j] > A[j + gap]; j -= gap)
            {
                  int temp = A[j];
                  A[j] = A[j + gap];
                  A[j + gap] = temp;
            }
      }
}
```

Грубый алгоритм поиска строки в тексте

Алгоритм проверяет на совпадение с искомой строкой фрагменты текста со всеми возможными смещениями.

1. Сравнить первый символ искомой строки с символом в тексте с текущим смещением относительно начала текста.
2. Если символы совпадают, то проверить следующие по порядку символы. Если совпадения нет – увеличить смещение строки в тексте на 1 и перейти на шаг 1.
3. Если все символы искомой строки совпадают с соответствующими символами в тексте, то поиск завершен успешно, иначе – сдвиги строки по тексту нужно продолжать до тех пор, пока «хвост» строки не выйдет за пределы текста, в этом случае завершение поиска безуспешно.

Алгоритм Рабина-Карпа для поиска строки в тексте

Использует контрольную сумму при поиске строки в тексте для того, чтобы сократить количество сравнений элементов строки и текста.

1. Вычислить контрольную сумму всех символов искомой строки.
2. Вычислить контрольную сумму символов подстроки текста начиная с заданного смещения и с длиной, равной длине искомой строки.
3. Если вычисленные суммы совпадают, то перейти к посимвольному сравнению строки и участка текста. Если суммы не совпали или посимвольное сравнение неуспешно – перейти к следующему шагу, иначе – поиск завершен успешно.
4. Из контрольной суммы подстроки текста вычесть значение первого символа текста, увеличить смещение подстроки в строке на 1, для этого к контрольной сумме подстроки текста прибавить последний символ подстроки текста после ее смещения.
5. Перейти к шагу 3 и продолжать до тех пор, пока последний символ текущей подстроки текста не совпадет с последним символом всего текста. В этом случае поиск безуспешен.

Алгоритм Боуера-Мура (упрощенный) для поиска строки в тексте

Для каждого символа «алфавита» текста и искомой строки вычислим возможное максимально возможное смещение строки по тексту и сохраним их в таблицу:

- Для символов текста, не входящих в искомую строку это смещение будет определяться значением -1;
- Для символов текста, входящих в искомую строку это смещение будет определяться расстоянием от правого конца искомой строки до первого данного символа в строке справа.

Основной алгоритм поиска:

1. Ставим искомую строку в очередную (начиная с первой) позицию текста и начинаем сравнивать ее с текстом, начиная с последнего символа строки и последовательно перебирая символы, двигаясь к началу строки и текста.
2. Когда будет обнаружено несовпадение очередного символа строки и текста – возьмем код символа текста и используем его в качестве индекса таблицы сдвигов. Достанем из таблицы возможный сдвиг по этому индексу и прибавим его к текущей позиции строки. Если же несовпадения обнаружено не будет (мы дойдем до начала строки) – значит, поиск завершился успешно.
3. Повторим шаги 1-2 до тех пор, пока строка не окажется сдвинутой до конца текста (поиск безуспешен).

Пример программного кода:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// функция находящая большее из двух чисел
unsigned int Max(unsigned int i, unsigned int j)
{
	if(i > j)
		return i;
	return j;
}
int main()
{
	unsigned int position = 0;  // Искомая позиция строки в тексте
	char T[] = "text with this string: abrakadabra (for example)";
	unsigned  int N = strlen(T); // Длина текста
	char S[] = "abrakadabra";
	unsigned  int M = strlen(S); // Длина искомой строки

	// Построим таблицу сдвигов:
	int SDVIGI[256];
	for (unsigned i = 0; i < 256; ++i)
		SDVIGI[i] = -1;
	// Для символов, не входящих в строку, сдвиг максимально возможный
	for (unsigned i = 0; i < M; ++i)
		SDVIGI[ S[i] ] = i;
	// Для символов, входящих в строку, сдвиг минимально
	// возможный (при повторах символов он будет уменьшен)

	// Алгоритм поиска Боуера-Мура:
	while(position <= (N - M))
	{
		int j = M - 1;
		while(j >= 0 && S[j] == T[position + j])
			j--; // Пока символы совпадают продвигаемся к началу искомой строки
		if (j < 0)
			break; // Мы нашли строку в тексте в данной позиции, прерываем цикл
		else
			position += Max(1, j - SDVIGI[T[position + j]]);
	}
	if (position <= (N - M))
		printf("String %s found at %u offset.\n", S, position+1);
	// Массив индексируется с нуля, поэтому прибавляем единицу,
	// чтобы выдать расстояние от начала текста
	else
		printf("String %s not found.\n", S);
	return 0;

}
```

Типовое задание на сортировку: написать и протестировать программу сортировки массива. Массив заполняется случайными числами при помощи функции **`rand()`**, после чего его нужно вывести на экран. Затем массив сортируется с подсчетом количества операций сравнения и перестановок элементов (отдельными переменными-счетчиками) и снова выводится на экран: отсортированный массив, его размерность, число сравнений, число перестановок.

Типовое задание на поиск подстроки: «текст» задается случайным целочисленным массивом. Выводим получившийся массив, чтобы его увидел пользователь. У пользователя запрашивается искомая подстрока, затем запрошенное ищется с подсчетом количества операций сравнения элементов и печатается позиция, в которой подстрока нашлась, а также количество операций сравнения элементов, которые были выполнены.

Варианты задач для решения

1. Вариант Реализовать приближение линейной функции **методом наименьших квадратов**.
Значения хранятся в вещественных массивах **`x`** и **`y`**, размер массивов задаётся с помощью **`#define`**.
Пользователь вводит значения из консоли. Вывести параметры прямой, являющейся наилучшим приближением к заданным точкам на плоскости.

2. Вариант
Сортировка массива действительных чисел методом вставки в порядке возрастания.
Массив заполняется случайными значениями с помощью функции rand(). Размер массива задаётся с помощью #define. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

3. Вариант
Сортировка массива действительных чисел **методом вставки** в порядке убывания.
Массив заполняется случайными значениями с помощью функции rand(). Размер массива задаётся с помощью #define. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

4. Вариант
Сортировка массива действительных чисел **методом выбора** в порядке возрастания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

5. Вариант
Сортировка массива действительных чисел **методом шейкера** в порядке убывания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

6. Вариант
Сортировка массива действительных чисел **методом выбора** в порядке убывания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

7. Вариант
Сортировка массива действительных чисел **методом пузырька** в порядке возрастания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

8. Вариант
Сортировка массива действительных чисел **методом пузырька** в порядке убывания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы.

9. Вариант
Выполнить транспонирование матрицы.
Размеры матрицы задаются с помощью **`#define`**. Матрица заполняется случайными значениями с помощью функции **`rand()`**. Вывести исходную и транспонированную матрицы, их размерности.

10. Вариант
Грубый алгоритм поиска **первого вхождения подмассива** в массиве.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задается с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сами массив и подмассив.

11. Вариант
Грубый алгоритм поиска **последнего вхождения подмассива** в массиве.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задается с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сами массив и подмассив.

12. Вариант

Поиск всех вхождений подмассива в массиве методом Рабина-Карпа.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задается с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождений подмассива, сами массив и подмассив.

13. Вариант

Грубый алгоритм поиска **первого вхождения подмассива** в массиве с одной ошибкой (то есть один элемент найденного вхождения имеет право отличаться от искомого массива).
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задается с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сам массив и искомый подмассив.

14. Вариант

Грубый алгоритм поиска **первого вхождения подмассива** в массиве с заданным пользователем числом ошибок (то есть указанное количество элементов найденного вхождения имеет право отличаться от искомого массива).
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задаётся с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сам массив и искомый подмассив.

15. Вариант

Грубый алгоритм поиска **последнего вхождения подмассива** в массиве с одной ошибкой (то есть один элемент найденного вхождения имеет право отличаться от искомого массива).
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задается с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сам массив и искомый подмассив.

16. Вариант

Поиск первого вхождения подмассива в массиве методом Боуера-Мура.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задаётся с помощью **`#define`**. Необходимо вывести количество сравнений, позицию начала вхождения подмассива, сам массив и искомый подмассив.

17. Вариант

Поиск всех вхождений подмассива в массиве методом Боуера-Мура.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива и подмассива задаётся с помощью **`#define`**. Необходимо вывести количество сравнений, позиции всех вхождений подмассива, сам массив и искомый подмассив.

18. Вариант

Построить **линейную интерполяцию или экстраполяцию** по заданной таблице значений функции. Даны два вещественных массива X и Y, значения в массиве X монотонно возрастают. У пользователя запрашивается значение x, для которого надо найти соответствующий интервал в массиве X: если введенное x значение меньше минимального значения в массиве X или больше максимального – то надо выполнить нахождение значения y для этого x с помощью линейной экстраполяции крайней пары точек. В противном случае нужно найти интервал, в который попадает введенное значение x, после чего нужно найти значение y для этого x с помощью линейной интерполяции.
Вывести исходные массивы, введенное значение x и найденное значение y.

19. Вариант

Вывести **строки матрицы** в порядке возрастания суммы их элементов.
Для этого завести отдельный массив сумм элементов по строкам, искать каждый раз минимальный элемент из оставшихся и выводить соответствующую строку матрицы. Матрица заполняется случайными значениями с помощью функции **`rand()`**. Размеры матрицы задаются с помощью **`#define`**. Вывести исходную матрицу, далее преобразованную матрицу, где строки должны идти в указанном порядке (то есть, необходимо поменять местами строки исходной матрицы).

20. Вариант

Вывести **строки матрицы** в порядке убывания суммы их элементов.
Для этого завести отдельный массив сумм элементов по строкам, искать каждый раз максимальный элемент из оставшихся и выводить соответствующую строку матрицы. Матрица заполняется случайными значениями с помощью функции **`rand()`**. Размеры матрицы задаются с помощью **`#define`**. Вывести исходную матрицу, далее преобразованную матрицу, где строки должны идти в указанном порядке (то есть, необходимо поменять местами строки исходной матрицы).

21. Вариант

Подсчитать **среднее значение и дисперсию** в каждом столбце матрицы.
Матрица заполняется случайными значениями с помощью функции **`rand()`**. Размеры матрицы задаются с помощью **`#define`**.

22. Вариант

Реализовать **сложение** трех десятичных чисел в столбик. Числа представлены массивами десятичных цифр и запрашиваются у пользователя.
Размеры массивов, представляющих слагаемые, задаются с помощью **`#define`**. Выводить слагаемые и результат. Учтите, что результат может быть длиннее каждого из слагаемых.

23. Вариант

Реализовать **вычитание** двух шестнадцатеричных чисел в столбик. Числа представлены массивами шестнадцатеричных цифр и запрашиваются у пользователя.
Размеры массивов, представляющих числа, задаются с помощью **`#define`**. Выводить слагаемые и результат.

24. Вариант

Построить и вывести **вертикальную гистограмму** (количество элементов каждого значения) значений целочисленного массива. Каждая строка гистограммы содержит столько символов «**`*`**», сколько раз встретилось уникальное значение входного массива, соответствующее данной строке гистограммы. Строки гистограммы должны выводиться от меньшего значения исходного массива к большему.
Размеры исходного массива и количество разных значений в гистограмме задать через **`#define`**.

25. Вариант

Построить **график функции**, фигурировавшей в уравнении, которое решалось на предыдущей задаче практикума, в окрестности найденного корня уравнения. График должен изображаться в виде матрицы символов, в которой значения функции помечаются символом «**`*`**», а также помечаются оси координат, прочие символы должны быть пробелами. Необходимо сделать преобразование масштаба, чтобы окрестность корня изображалась максимально информативно.
Размерность матрицы задается с помощью **`#define`** (не более 80 столбцов и 25 строк).

26. Вариант

Сортировка массива действительных чисел **методом Шелла** в порядке убывания.
Массив заполняется случайными значениями с помощью функции **`rand()`**. Размер массива задаётся с помощью **`#define`**. Необходимо вывести количество перестановок и сравнений, а также входной и выходной массивы. Если останется время – сравнить количество перестановок и сравнений с методом сортировки вставками.

27. Вариант

Написать программу, которая объединяет два упорядоченных по возрастанию массива в один, также упорядоченный массив и проверяет, есть ли в выходном массиве пара соседних элементов, сумма которых равна заданному числу.
Размеры массивов задаются с помощью **`#define`**. Нужно распечатать исходные массивы и результат работы программы.

28. Вариант

Написать программу, удаляющую из вещественного массива все элементы, которые отклоняются от среднего значения более, чем на стандартное отклонение.
Размер исходного массива задаётся с помощью **`#define`**. Результат сохраняется в этом же массиве, количество оставшихся элементов запоминается в переменной N. Нужно распечатать исходный массив, среднее значение, стандартное отклонение и получившийся прореженный массив.

29. Вариант

Написать программу, которая приводит заданную квадратную матрицу к треугольному виду по методу Гаусса (домножение строк на множитель и сложение двух соседних строк с заменой второй строки на результат сложения).
Матрица заполняется случайными значениями с помощью функции **`rand()`**. Размеры матрицы задаются с помощью **`#define`**.

30. Вариант

В исходном случайном целочисленном массиве оставить только уникальные значения. Для эффективного решения этой задачи его следует отсортировать любым способом, после чего просканировать для удаления рядом стоящих повторов и подсчета количества этих повторов.
Размер исходного массива задаётся с помощью **`#define`**. Результат сохраняется в этом же массиве, его уменьшенная длина запоминается в переменной N. Нужно распечатать исходный массив, и результат работы программы: для каждого уникального значения выходного массива распечатать количество его повторов.

Глава 4. Функции.

Вычисление значений математических функций

Как вы знаете, любую математическую функцию можно разложить в ряд Тейлора.

Рассмотрим для примера функцию экспоненты

$$e^x = \sum_{i=0}^{\infty} a_i = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

Т.е. $a_i=x^i/i!$. Учтите, что значение a_i не нужно вычислять каждый раз. Необходимо выразить значение $(i+1)$-го члена последовательности через i-й:

$$a_{i+1} = \frac{x^{i+1}}{(i+1)!} = \frac{x * x^i}{i! * (i+1)} = a_i \frac{x}{i+1}$$

Теперь напишем функцию, вычисляющую значение с заданной точностью *eps*:

```
double f(double x,double eps)
{
    double s=0;// Тут копим сумму
    double ai=1.; // Тут храним i-ый член ряда
    int i=0; // начальное значение i=0
    while(fabs(ai)>eps)
// Суммировать будем пока член ряда an не станет достаточно маленьким
    {
        s+=ai; // суммируем очередной член ряда
        ai*=x/(i+1); // пересчитываем a(i+1) через a(i)
        i++; // переходим к следующему члену
    }
    return s; // получившаяся сумма
}
```

Программа для тестирования этой функции будет выглядеть так:

```
int main()
{
    system("chcp 1251 > NUL");
    double xb,xe,step,eps;
    printf("x начальное=");
    scanf("%lf",&xb);
    printf("x конечное=");
    scanf("%lf",&xe);
    printf("шаг=");
    scanf("%lf",&step);
    printf("точность=");
    scanf("%lf",&eps);
    getchar();
    for(double x=xb;x<xe+step/2.;x+=step)
        printf("%lf\t%lf\t%lf\n",x, f(x,eps), exp(x));
    getchar();
    return 0;
}
```

Т.е. в программе вводятся *x* начальное и *x* конечное, шаг, с которым нужно получать значения и точность вычисления *eps*. И далее в цикле печатаются значения *x*, получившейся функции и библиотечной функции.

Обратите внимание на условие `x<xe+step/2.` Нюанс в том, что если написать `x<=xe`, то последняя строка может и не напечататься. Причина в том, что двоичные числа с плавающей запятой дают приближенное значение при переводе в десятичную систему счисления. И, например, вместо 1.1 может получиться 1.0999999999999999.

Варианты задач для решения.

Напишите функцию `double f(double x,double eps)` вычисляющую значение библиотечной функции, заданной разложением в ряд Тейлора $f(x) = \sum_{n=0}^{\infty} a_n(x)$. Суммирование завершайте по

достижению условия $|a_n(x)| \le eps$. Для вычисления a_n воспользуйтесь рекуррентными соотношениями. Функция должна поверять значение *x* и выдавать сообщение об ошибке при задании *x* вне диапазона, разрешённого для используемого ряда.

Функцию поместить в отдельный файл (например, *lib.cpp*). Объявление функции поместить в заголовочный файл (например, *lib.h*). Программу тестирования этой функции поместить в другой файл (например, *task4.cpp*). В ней нужно организовать ввод параметра *eps*, интервал и шаг изменения аргумента *x* и печать таблицы значений аргумента, функции $f(x)$ и соответствующей библиотечной функции.

1. Вариант $f(x)=\cos(x)=\sum_{n=0}^{\infty} a_n(x)$, $a_n=\frac{(-1)^n (x)^{2n}}{(2n)!}$
2. Вариант $f(x)=arctg(x)=\frac{\pi}{2}+\sum_{n=0}^{\infty} a_n(x)$, $a_n=\frac{(-1)^{n+1} x^{-2n-1}}{2n+1}$, $\|x\|>1$
3. Вариант $f(x)=(1-x)^m=1+\sum_{n=1}^{\infty} a_n(x)$, $a_n=\frac{(-1)^n m(m-1)...(m-n+1)(x)^n}{(n)!}$, $\|x\|<1$,m=1/2
4. Вариант $f(x)=\mathrm{sinc}(x)=\sum_{n=0}^{\infty} a_n(x)$, $a_n=\frac{(-1)^n (x)^{2n}}{(2n+1)!}$
5. Вариант $f(x)=arcch(x)=\ln(2x)-\sum_{n=1}^{\infty}\frac{1*3*5*...*(2n-1)(x)^{-2n}}{2*4*6*...*(2n)}$
6. Вариант $f(x)=arctg(x)=\sum_{n=0}^{\infty} a_n(x)$, $a_n=\frac{(-1)^n x^{2n+1}}{2n+1}$, $\|x\|<1$
7. Вариант $f(x)=sh(x)=\sum_{n=1}^{\infty} a_n(x)$, $a_n=\frac{x^{2n-1}}{(2n-1)!}$
8. Вариант $f(x)=\ln(x)=\sum_{n=1}^{\infty}\frac{(x-1)^n}{nx^n}$, x > 0.5
9. Вариант $f(x)=arcctg(x)=\frac{\pi}{2}+\sum_{n=0}^{\infty} a_n(x)$, $a_n=\frac{(-1)^{n+1} x^{2n+1}}{2n+1}$, $\|x\|>1$
10. Вариант $f(x)=\frac{1}{1-x}=\sum_{n=0}^{\infty}\frac{(x-x_0)^n}{(1-x_0)^{n+1}}$
11. Вариант $f(x)=\sin(x)=\sum_{n=1}^{\infty}\frac{(-1)^{n-1}(x)^{2n-1}}{(2n-1)!}$
12. Вариант

$$f(x)=arccth(x)=\sum_{n=0}^{\infty}\frac{(x)^{-2n-1}}{2n+1}$$, |x|>1

13. Вариант

$$f(x)=\ln(x)=2\sum_{n=0}^{\infty}\frac{(x-1)^{2n+1}}{(2n+1)(x+1)^{2n+1}}$$, x>0

14. Вариант

$$f(x)=\ln(x+1)=\sum_{n=1}^{\infty}\frac{(-1)^{n+1}(x)^{n}}{n}$$, |x|<1

15. Вариант

$$f(x)=(1+x)^{m}=1+\sum_{n=1}^{\infty}a_n(x)\quad a_n=\frac{m(m-1)...(m-n+1)(x)^{n}}{(n)!}$$, |x|<1,m=1/3

16. Вариант

$$f(x)=\ln(1-x)=-\sum_{n=1}^{\infty}\frac{(x)^{n}}{n}$$, |x|>1

17. Вариант

$$f(x)=\ln\left(\frac{x+1}{x-1}\right)=2\sum_{n=0}^{\infty}\frac{(x)^{-2n-1}}{2n+1}$$, |x|>1

18. Вариант

$$f(x)=\arcsin(x)=x+\sum_{n=1}^{\infty}\frac{1*3*5*...*(2n-1)(x)^{2n+1}}{2*4*6*...*(2n)*(2n+1)}$$

19. Вариант

$$f(x)=\arccos(x)=\frac{\pi}{2}-x-\sum_{n=1}^{\infty}\frac{1*3*5*...*(2n-1)(x)^{2n+1}}{2*4*6*...*(2n)*(2n+1)}$$

20. Вариант

$$f(x)=\ln\left(\frac{1+x}{1-x}\right)=2\sum_{n=0}^{\infty}\frac{(x)^{2n+1}}{2n+1}$$, |x|<1

21. Вариант

$$f(x)=(1-x)^{-m}=1+\sum_{n=1}^{\infty}a_n(x)\quad a_n=\frac{m(m+1)...(m+n-1)(x)^{n}}{(n)!}$$, |x|<1, m=1/4

22. Вариант

$$f(x)=arcsh(x)=x+\sum_{n=0}^{\infty}a_n(x)=x+\sum_{n=1}^{\infty}(-1)^{n}\frac{1*3*5*...*(2n-1)(x)^{2n+1}}{2*4*6*...*(2n)*(2n+1)}$$

23. Вариант

$$f(x)=arcth(x)=\sum_{n=0}^{\infty}a_n(x)=\sum_{n=0}^{\infty}\frac{(x)^{2n+1}}{2n+1}$$, |x|<1

24. Вариант

$$f(x)=ch(x)=\sum_{n=0}^{\infty}a_n(x)\quad a_n=\frac{x^{2n}}{(2n)!}$$

25. Вариант

$$f(x)=x\cos(3x)=\sum_{n=0}^{\infty}a_n(x)=\sum_{n=1}^{\infty}\frac{(-1)^n 3^{2n}(x)^{2n+1}}{(2n)!}$$

26. Вариант

$$f(x)=\ln(1-x^2)=-\sum_{n=0}^{\infty}a_n(x)=-\sum_{n=1}^{\infty}\frac{(x)^{2n}}{n}$$

27. Вариант

$$f(x)=0.5\ln\left(\frac{1+x}{1-x}\right)=\sum_{n=0}^{\infty}a_n(x),\ a_n=\frac{x^{2n+1}}{2n+1}$$

28. Вариант

$$f(x)=0.25\ln\left(\frac{1+x}{1-x}\right)-0.5*arctg(x)=\sum_{n=0}^{\infty}a_n(x),\ a_n=\frac{x^{4n-1}}{4n-1}$$

29. Вариант

$$f(x)=\frac{\sin(x)}{x}=\sum_{n=0}^{\infty}a_n(x),\quad a_n=(-1)^n\frac{x^{2n}}{(2n+1)!}$$

30. Вариант

$$f(x)=\frac{\cos(\sqrt{x})-1}{x}=\sum_{n=0}^{\infty}a_n(x)=\sum_{n=1}^{\infty}\frac{(-1)^n(x)^{n-1}}{(2n)!}$$

Вычисление определенных интегралов

Для вычисления определенного интеграла $\int_a^b f(x)dx$ можно использовать один из следующих численных методов, различающихся по своей точности, достигаемой при одинаковых затратах вычислительных ресурсов.

Методы нижних, верхних и средних прямоугольников

Будем рассматривать определенный интеграл как площадь под графиком подынтегральной функции. Разобьем весь интервал интегрирования на **N** отрезков одинаковой длины. Тогда аппроксимацией для площади под графиком функции может служить сумма площадей прямоугольников, ширина которых равна длине одного отрезка разбиения, а высота – значению функции **f(x)** в данном месте графика функции. При этом метод нижних прямоугольников будет приближать значение интеграла снизу:

$I = \sum\Delta F_i,\ \Delta F_i = h \cdot \min(f(x_i), f(x_i+h)),$ где $h = x_{i+1} - x_i$

Метод верхних прямоугольников будет приближать значение интеграла сверху:

$I = \sum\Delta F_i,\ \Delta F_i = h \cdot \max(f(x_i), f(x_i+h)),$ где $h = x_{i+1} - x_i$

Нужно обратить внимание, что методы нижних и верхних прямоугольников не тождественны методам правых и левых прямоугольников, так как функции на различных участках могут как возрастать, так и убывать. Также нужно принимать во внимание то, что истинное значение интеграла будет лежать строго между результатами, посчитанными методами нижних и верхних прямоугольников, то есть, вычисляя два этих метода одновременно – можно оценить и точность получившегося решения. При этом затраты вычислительных ресурсов не сильно вырастут, если устранить повторные вычисления функции в одной и той же точке.

Метод средних прямоугольников будет давать результаты, средние между описанными выше двумя методами:

$I = \sum\Delta F_i,\ \Delta F_i = h \cdot f(x_i + h/2),$ где $h = x_{i+1} - x_i$

Метод трапеций

Метод трапеций – чуть точнее метода прямоугольников, он заключается в том, что вычисляется площадь трапеции, а не прямоугольника:

$I = \sum\Delta F_i,\ \Delta F_i = h \cdot (f(x_i)+ f(x_i+h)) / 2,$ где $h = x_{i+1} - x_i$

Нужно обратить внимание, что этот метод не совпадает с методом средних прямоугольников, хотя и похож на него.

Метод парабол (метод Симпсона)

Большой выигрыш в точности вычислений при эквивалентной затрате вычислительных ресурсов дает метод Симпсона, при котором на каждом сегменте разбиения находятся коэффициенты параболы по трем точкам, находящимся на границах сегмента и в его центре, интеграл от квадратичной функции легко вычисляется аналитически, что дает в результате следующую формулу для метода парабол:

$$I = \sum \Delta F_i,\ \Delta F_i = h \cdot (f(x_i) + 4 \cdot f(x_i + h/2) + f(x_i + h)) / 6,\ \text{где}\ h = x_{i+1} - x_i$$

Рассмотрим функцию, вычисляющую значение интеграла этим способом:

```
#include <math.h>
// наша интегрируемая функция:
double f(double x)
{
	return cos(x);
}
// первообразная функция:
double Int(double x)
{
	return sin(x);
}
// функция вычисляющая интеграл на интервале [a,b]
// по формуле Симпсона с заданной точностью eps
int N_Iter; // глобальная переменная для количества итераций
double Integral(double a,double b,double eps)
{
	int k=10; // начальное к-во разбиений
	N_Iter=0;
	// значения интегралов на текущей и предыдущей интерациях:
	double val1,val2=0.;
	do
	{
		val1=val2; // предыдущее значение равно текущему
		double s=0; // переменная для суммирования
		double x=a; // начинаем с a
		double h=(b-a)/k; // шаг интегрирования
		for(int i=0;i<k;i++)// повторяем k раз
		{
			double t=h*(f(x)+4.*f(x+h/2)+f(x+h))/6.;
			s+=t; // суммируем очередную площадь
			x+=h; // переходим к следующему интервалу
		}
		val2 = s; // вычисленное новое значение
		k*=2; // удвоение количества разбиений
		N_Iter++; // подсчитываем к-во итераций
	}while (fabs(val1-val2)>eps); // повторяем до достижения точности eps
	return val2; // возвращаем результат интегрирования
}
```

Проинтегрируем ее в интервале от $-\pi/2$ до $\pi/2$ с точностью eps:

```
#include <stdio.h>
#include <stdlib.h>
#define _USE_MATH_DEFINES
#include <math.h>
extern int N_Part; // переменная N_Part объявлена в другом файле
double Integral(double a,double b,double eps);
double Int(double x);
// ---------------------------------------------
extern int N_Iter;
int main()
{
	system("chcp 1251 > NUL");
	double eps=1e-6; // точность интегрирования
	double xb=-M_PI_2; // начало интервала
	double xe=M_PI_2; // конец интервала
	double res=Integral(xb,xe,eps); // вычисление нового значения
	printf("Вычисленное значение=%lf\n",res);
	printf("Аналитическое значение=%lf\n",Int(xe)-Int(xb));
```

```
        printf("Количество итераций=%d\n",N_Iter);
        getchar();
        return 0;
}
```

Варианты задач для решения.

Вычислить значение определённого интеграла двумя способами. Для вычисления необходимо задать границы интегрирования и точность расчетов в функции *main*(). Их можно либо ввести с клавиатуры, либо передать через параметры командной строки. Функции *f*() (подынтегральная функция), *fint*() (аналитическое вычисление интеграла), *Integral*() (интеграл, посчитанный численно) нужно перенести в отдельный *.cpp файл. А их объявления – в заголовочный файл.

Функции *Integral*() передаётся начало и конец интервала интегрирования и требуемая точность. Она должна возвращать вычисленное значение интеграла и записывать финальное количество разбиений в глобальную переменную.

В функции *main*() нужно найти и вывести

- Значение интеграла, полученное численным методом;
- Аналитическое значение интеграла;
- Достигнутая точность;
- Количество итераций (удвоений разбиения), которые понадобились для достижения требуемой точности.

Вид подынтегральной функции и способы вычисления для каждого варианта приведены в таблице.

Вар.	**Подынтегральная функция**	**Первообразная**	**Метод 1**	**Метод 2**
1	$\sqrt{\frac{1}{1-x^2}}$	$\arcsin(x)$	Верхних прямоугольников	Трапеций
2	$\frac{2}{x}$	$ln(x^2)$	Верхних прямоугольников	Нижних прямоугольников
3	$2x\cos(2x)+\sin(2x)$	$x\sin(2x)$	Трапеций	Симпсона
4	$ln(x)+1$	$x*ln(x)$	Трапеций	Верхних прямоугольников
5	$(3+5x)^3$	$\frac{(3+5x)^4}{20}$	Трапеций	Верхних прямоугольников
6	$\frac{2x}{1-x}+\frac{x^2}{1-x^2}$	$\frac{x^2}{1-x}$	Трапеций	Нижних прямоугольников
7	$\frac{1}{x^2-1}$	$\frac{1}{2}\ln\left\|\frac{x-1}{x+1}\right\|$	Верхних прямоугольников	Средних прямоугольников
8	$\frac{1}{1-x^2}$	$\frac{1}{2}\ln\left\|\frac{1+x}{1-x}\right\|$	Трапеций	Нижних прямоугольников
9	$\sqrt{1-x^2}$	$\frac{x}{2}\sqrt{1-x^2}+\frac{\arcsin(x)}{2}$	Верхних прямоугольников	Нижних прямоугольников
10	$\sin(2x)$	$\sin^2(x)$	Трапеций	Симпсона

11	$\mathrm{tg}(x)$	$-\log\lvert\cos(x)\rvert$	Нижних прямоугольников	Симпсона
12	$\frac{1}{1+x^2}$	$\mathrm{arctg}(x)$	Трапеций	Нижних прямоугольников
13	$\frac{1}{\sin(x)}$	$\log\left\lvert\mathrm{tg}\left(\frac{x}{2}\right)\right\rvert$	Верхних прямоугольников	Нижних прямоугольников
14	$\frac{1}{\sqrt{1+x^2}}$	$\log\left\lvert x+\sqrt{1+x^2}\right\rvert$	Средних прямоугольников	Симпсона
15	$\frac{x}{1+x^2}$	$\frac{\log\lvert x^2+1\rvert}{2}$	Нижних прямоугольников	Трапеций
16	$\frac{\sin(x)}{(1+\cos(x))^2}$	$\frac{1}{1+\cos(x)}$	Трапеций	Нижних прямоугольников
17	$x\,e^{x^2}$	$\frac{e^{x^2}}{2}$	Трапеций	Нижних прямоугольников
18	$e^x(\cos(x)-\sin(x))$	$e^x\cos(x)$	Верхних прямоугольников	Нижних прямоугольников
19	$\frac{1}{\ln(x)}-\frac{1}{\ln^2(x)}$	$\frac{x}{\ln(x)}$	Трапеций	Верхних прямоугольников
20	$x(2\ln(x)+1)$	$x^2\ln(x)$	Трапеций	Средних прямоугольников
21	$\frac{e^x-e^{-x}}{2}$	$\frac{e^x+e^{-x}}{2}$	Трапеций	Нижних прямоугольников
22	$1-\frac{e^x-e^{-x}}{(e^x+e^{-x})^2}$	$\frac{e^x-e^{-x}}{e^x+e^{-x}}$	Трапеций	Симпсона
23	$\frac{1}{x^2}\sin\left(\frac{1}{x}\right)$	$\cos\left(\frac{1}{x}\right)$	Трапеций	Нижних прямоугольников
24	$\frac{1}{\sin(x)}$	$\ln\left\lvert tg\frac{x}{2}\right\rvert$	Верхних прямоугольников	Нижних прямоугольников
25	$\frac{1}{\cos^2(x)}$	$tg(x)$	Нижних прямоугольников	Симпсона
26	$\frac{x}{x^2+1}$	$\frac{\ln\lvert x^2+1\rvert}{2}$	Трапеций	Средних прямоугольников
27	$\mathrm{tg}^2(x)$	$\mathrm{tg}(x)-x$	Нижних	Верхних

			прямоугольников	прямоугольников
28	$3x^2-4x$	x^3-2x^2	Верхних прямоугольников	Трапеций
29	$\sqrt{x^2+1}$	$\frac{x}{2}\sqrt{x^2+1}-\frac{\ln\left\|x+\sqrt{x^2+1}\right\|}{2}$	Симпсона	Нижних прямоугольников
30	$\sqrt{x^2-1}$	$\frac{x}{2}\sqrt{x^2-1}-\frac{\ln\left\|x+\sqrt{x^2-1}\right\|}{2}$	Трапеций	Средних прямоугольников

Глава 5. Коллоквиум, методика проведения и типовые вопросы.

Коллоквиум проводится на семинарском занятии в середине семестра (обычно – до 1-го ноября) в форме тотального опроса с билетами.

Билеты

1. Вариант 1) Основы синтаксиса языка Си. Ключевые слова. Фундаментальные типы данных. Определение переменных и констант. Выражения, операции, комментарии. 2) Разные виды цикла while, do.
2. Вариант 1) Операторы в выражениях языка Си. Приоритет операторов. Оператор sizeof(). 2) Директива #define препроцессора и ее использование. Макроопределения с параметром.
3. Вариант 1) Управляющий оператор switch. 2) Директивы условной компиляции препроцессора и их использование.
4. Вариант 1) Управляющие операторы if, goto. 2) Модульный подход в программировании. Использование *.h файлов. Раздельная компиляция.
5. Вариант 1) Массивы. Передача массивов в функции. 2) Локальные, глобальные, статические переменные.
6. Вариант 1) Основные действия, выполняемые при помощи отладчика в среде Microsoft Visual Studio. 2) Цикл for.
7. Вариант 1) Функция main(), ее параметры. 2) Оператор typedef. Приведение типов.
8. Вариант 1) Использование функций: заголовок, тело и вызов функции. 2) Управляющие операторы if, goto.
9. Вариант 1) Статические функции. 2) Блоки и правила видимости переменных.
10. Вариант 1) Передача параметров в функции. Передача параметров по значению. 2) Операторы в выражениях языка Си. Приоритет операторов. Оператор sizeof().
11. Вариант 1) Работа в среде Microsoft Visual Studio: создание проекта. Основные файлы проекта. Этапы создания программы: компиляция, линковка, запуск на выполнение, отладка. 2) Цикл for.
12. Вариант 1) Разные виды цикла while, do. 2) Массивы. Передача массивов в функции.
13. Вариант 1) Директива #include препроцессора и ее использование. 2) Функция main(), ее параметры.
14. Вариант 1) Процедурный подход программирования. Определение функции. Прототип функции. 2) Операторы инкремента и декремента.
15. Вариант 1) Передача параметров в функции. Передача параметров по значению. 2) Директивы условной компиляции препроцессора и их использование.

16. Вариант 1) Локальные, глобальные, статические переменные. 2) Использование функций: заголовок, тело и вызов функции.
17. Вариант 1) Директива #define препроцессора и ее использование. Макроопределения с параметром. 2) Управляющий оператор switch.
18. Вариант 1) Разные виды цикла while, do. 2) Директива #include препроцессора и ее использование.
19. Вариант 1) Работа в среде Microsoft Visual Studio: создание проекта. Основные файлы проекта. Этапы создания программы: компиляция, линковка, запуск на выполнение, отладка. 2) Глобальные и внешние переменные.
20. Вариант 1) Виды операторов присваивания в языке Си. 2) Оператор typedef. Приведение типов.
21. Вариант 1) Операторы инкремента и декремента. 2) Рекурсивный вызов функций.
22. Вариант 1) Операторы в выражениях языка Си. Приоритет операторов. Оператор sizeof(). 2) Статические переменные.
23. Вариант 1) Основные действия, выполняемые при помощи отладчика в среде Microsoft Visual Studio. 2) Массивы: одномерные и двумерные.
24. Вариант 1) Разные виды цикла while, do. 2) Модульный подход в программировании. Использование *.h файлов. Раздельная компиляция.
25. Вариант 1) Логические (булевские) операторы и операторы сравнения. 2) Цикл for.
26. Вариант 1) Массивы. Передача массивов в функции. 2) Локальные, глобальные, статические переменные.
27. Вариант 1) Основные действия, выполняемые при помощи отладчика в среде Microsoft Visual Studio. 2) Массивы: одномерные и двумерные.
28. Вариант 1) Функция main(), ее параметры. 2) Статические переменные.
29. Вариант 1) Операторы инкремента и декремента. 2) Управляющие операторы if, goto.
30. Вариант 1) Статические функции. 2) Управляющий оператор switch.

Факультативные задания

Так как коллоквиум проводится на семинарском занятии, то у студентов появляются дополнительные 2 недели, которые следует использовать для доделки и сдачи всех выданных ранее задач. Практика показывает, что к этому моменту «хвосты» имеет, как минимум, половина группы.

Студенты, которые успешно сдали все предыдущие задания, могут получить факультативное задание. Учет этих факультативных задач отличается от учета обычных заданий: они не являются обязательными, то есть, отсутствие их решения – не ухудшает оценки за успеваемость студента в течение семестра, в то же

время, наличие такой сданной задачи улучшает оценку студента. Если студент не успевает их сдать до шестого семинара, то такому студенту рекомендуется сначала сдавать плановые, обычные задания и лишь когда плановые задания сданы – вернуться к выполнению факультативных заданий.

Во всех приведенных ниже заданиях нужно оформить основную часть задания в виде функции, которой передаются все необходимые аргументы и которая возвращает результат. Весь ввод-вывод сосредоточить в функции `main()`.

1. Вариант Написать программу, вычисляющую наибольший общий делитель для двух введенных чисел. Алгоритм Евклида для нахождения наибольшего общего делителя двух чисел выглядит следующим образом. Даны два числа:a и b. Пока они не равны, вычитать из большего числа меньшее. Оставшееся значение a (или b) и будет наибольшим общим делителем двух чисел.

2. Вариант Лягушка изначально сидит в точке 0 числовой прямой. Каждую секунду она прыгает на 1 вправо, пока не достигнет точки K. Затем она начинает каждую секунду прыгать на 1 влево, пока не вернется в точку 0, затем – опять вправо и т. д. Написать программу определяющую, где окажется лягушка через T секунд. Значения K и T ввести с клавиатуры, если их нет в параметрах командной строки.

3. Вариант

Написать программу, вычисляющую определитель квадратной матрицы произвольной размерности.

Размерность и коэффициенты матрицы (не более 10) запрашиваются у пользователя, программа должна напечатать матрицу и ее определитель.

4. Вариант

Написать программу, выполняющую символьное умножение «в столбик» двух шестнадцатеричных чисел, вводимых пользователем «цифра за цифрой».

В программе использовать таблицу умножения для шестнадцатеричных цифр и правила умножения «в столбик». Не использовать приведение шестнадцатеричного числа к десятичной системе счисления.

5. Вариант

Написать программу, которая из данных n точек на плоскости находит все тройки точек, которые образуют равносторонние треугольники. Значения координат точек генерируются случайным образом в диапазоне [a, b], значения n, a, b вводятся пользователем.

6. Вариант

Отсортировать массив из уникальных элементов алгоритмом QuickSort: выбрать случайный элемент из середины массива, двигаясь от начала массива остановиться на элементе, который больше выбранного, двигаясь от конца массива остановиться на элементе, который меньше выбранного. Обменять найденные два элемента. Продолжить движение с обоих концов массива к его центру. Получив разбиение массива, в котором все элементы до некоторого – меньше выбранного, а все элементы после него – больше выбранного, повторить алгоритм рекурсивно для каждой из двух частей разбиения.

Сравнить количество операций сравнения и присваивания элементов массива с аналогичными показателями пузырьковой сортировки.

7. Вариант

Вычислить число π методом Монте-Карло: взять круг, вписанный в квадрат со стороной 10. Сгенерировать 1000 пар случайных чисел, каждое в диапазоне от 0 до 10. Каждую пару чисел считать `x-` и `y-` координатой некоторой точки внутри заданного квадрата. Для каждой такой точки определить – лежит ли она внутри круга? Отношение числа точек, попавших в круг к общему числу точек оценивает величину π / 4.

Сравнивая результат со встроенной константой `M_PI` оценить погрешность этого метода в зависимости от выбранного количества пар случайных чисел.

8. Вариант

Построить свой собственный генератор случайных чисел, основанный на линейном конгруэнтном методе: $x_{i+1} = (ax_i + c) \% M$ (a < M, c < M). Построить (в виде символьной матрицы 20*20) гистограмму достаточно большой сгенерированной последовательности чисел.

По гистограмме «на глаз» оценить равномерность распределения в зависимости от разных значений параметров `a, c, M`.

9. Вариант

Написать рекурсивную версию алгоритма, переводящего заданное число в двоичную систему счисления.

10. Вариант

Написать рекурсивную версию алгоритма, распечатывающего все простые множители заданного числа.

11. Вариант

В декартовой системе координат на плоскости в виде массива заданы координаты вершин треугольника. Определить, принадлежит ли точка с координатами (x,y) этому треугольнику.

12. Вариант

Даны N целых чисел X_1, X_2, ..., X_N. Требуется вычеркнуть из них минимальное количество чисел так, чтобы оставшиеся шли в порядке возрастания.

13. Вариант

Задан вес E пустой копилки и вес F копилки с монетами. В копилке могут находиться монеты N видов, для каждого вида известна ценность Pi и вес Wi одной монеты. Найти минимальную и максимальную суммы денег, которые могут находиться в копилке.

14. Вариант

Число называется совершенным, если оно равно сумме всех своих делителей, меньших его самого. Требуется найти все совершенные числа от M до N.

15. Вариант

Вывести все представления натурального числа N суммой натуральных чисел. Перестановка слагаемых нового способа представления не даёт.

16. Вариант

Многоугольник на плоскости задан целочисленными координатами своих N вершин в декартовой системе координат. Требуется найти площадь многоугольника. Стороны многоугольника не соприкасаются (за исключением соседних - в вершинах) и не пересекаются.

17. Вариант

Дано целое неотрицательное число в I-ричной системе счисления. Вывести это число в J-ричной системе счисления

18. Вариант

Дано N прямоугольников со сторонами, параллельными осям координат. Требуется определить площадь фигуры, образованной объединением данных прямоугольников

19. Вариант

Вывести в порядке возрастания все несократимые дроби, заключённые между 0 и 1, знаменатели которых не превышают N.

20. Вариант .

Дано N отрезков провода длиной L_1, L_2, ..., L_N сантиметров. Требуется с помощью разрезания получить из них K равных отрезков как можно большей длины, выражающейся целым числом сантиметров.

21. Вариант

По координатам вершин многоугольника требуется найти координаты его центра тяжести. Стороны многоугольника друг с другом не соприкасаются (за исключением соседних - в вершинах) и не пересекаются.

22. Вариант

Натуральное число представлено массивом десятичных цифр произвольной длины. Реализовать умножение «в столбик» двух таких чисел.

23. Вариант

Натуральное число представлено массивом десятичных цифр произвольной длины. Реализовать деление «в столбик» двух таких чисел, с выдачей результата в виде десятичной дроби (целая и дробные части отдельно) с заданной точностью.

24. Вариант

Дана последовательность целых чисел, удовлетворяющая условию Фибоначчи
$F_k = F_{k-1} + F_{k-2}$ для любого k>2.
Найти n-й член последовательности, если известны i-й и i+1-й члены, *i* может быть как больше, так и меньше *n*.

25. Вариант

В таблице из N строк и N столбцов клетки заполнены цифрами от 0 до 9. Требуется найти такой путь из клетки (1, 1) в клетку (N, N), чтобы сумма цифр в клетках, через которые он пролегает, была минимальной; из любой клетки ходить можно только вниз или вправо.
Выводятся *N* строк по *N* символов. Символ решётка показывает, что маршрут проходит через эту клетку, а минус - что не проходит. Если путей с минимальной суммой цифр несколько, вывести любой.

26. Вариант

Когда некоторая функция задана таблицей своих значений в большом количестве точек, то классическая интерполяция полиномами Лагранжа (см. курс лекций) будет давать неважные результаты. Вместо этого часто используют интерполяцию значений такой функции с помощью сплайнов: на каждом i-том интервале таблицы будем приближать заданную функцию полиномом порядка k (назовем его Sk(i)) так, чтобы выполнялось равенство значений двух соседних полиномов в каждом i-том узле таблицы Sk(i-1) = Sk(i) и значений всех производных этих двух соседних полиномов в этом же узле таблицы.
Набор этих условий дает однозначный алгоритм вычисления коэффициентов всех полиномов для всех интервалов, если на краях (то есть, в узлах таблицы 0 и N-1) мы будем полагать значения самых старших производных равными нулю. Для k=1 мы получим обычную кусочно-линейную интерполяцию, для больших k мы получим весьма гладкую, красиво выглядящую кривую с минимальными изгибами.
Постройте для заданной произвольной таблицы пар точек xi yi интерполяцию сплайнами второго, а если сможете – то и третьего порядка и сравните ее работу с интерполяцией полиномом Лагранжа.

27. Вариант

Для функции y=sin(1/x) в ближней окрестности нуля сложно применить обычные методы вычисления определенного интеграла, так как она очень быстро меняется и методы, использующие разбиение на равные интервалы будут работать плохо.

Попробуйте реализовать подсчет такого определенного интеграла, разложив эту функцию в ряд Тейлора и вычисляя сумму определенных интегралов для каждого члена этого разложения до достижения заданной точности. Точность можно контролировать как модуль разности между двумя последовательными приближениями значения вычисляемого интеграла.

28. Вариант

Максимально быстро найти все возможные корни уравнения $\cos(x) + \cos(x^2) - 1.55 = x$

Проблема в том, что метод касательных будет гарантировать сходимость в довольно узкой окрестности каждого корня, поэтому нужно использовать комбинацию метода дихотомии или хорд (чтобы приблизиться к корню) и переход к методу касательных, когда это становится возможно (т.е. выполняется условие сходимости).

29. Вариант

Алгоритм быстрой сортировки имеет оценку времени его работы пропорциональный $N*\log_2(N)$. Построим алгоритм сортировки натуральных чисел, который будет работать за константное время. Учтем, что в компьютерном представлении все натуральные числа имеют одно и то же количество двоичных разрядов (скажем, 32). Организуем два прохода нашего массива, движущихся навстречу друг другу, как в алгоритме QuickSort: только первый проход (идущий от начала массива) будет обнаруживать единицу в старшем разряде числа, а второй (идущий навстречу от конца массива) будет обнаруживать ноль в этом же старшем разряде числа. Как только такая пара чисел будет обнаружена – поменяем их местами и продолжим эти проходы. В конце проходы сойдутся где-то в середине массива, при этом все числа слева от этой точки будут иметь в старшем разряде ноль, а все числа справа – единицу. Дальше применим этот же алгоритм рекурсивно к каждой из двух получившихся частей массива для второго по старшинству двоичного разряда, потом для третьего и так далее. Таким образом в самом конце и получим полностью отсортированный массив. Оценка производительности при этом получится 32*N, так как количество разрядов числа – 32 и на каждой итерации мы перебираем все элементы массива. Очевидно, что при очень больших N этот результат будет эффективнее быстрой сортировки.

30. Вариант

Дан большой массив произвольных натуральных чисел (без повторов), нужно максимально быстро отвечать на вопрос: есть в нем заданное значение или нет? Если его отсортировать и потом использовать для поиска метод деления пополам, то время такого поиска будет пропорционально $\log_2(N)$.

Построим другой алгоритм, который нам обеспечит константное время поиска: заведем вспомогательный массив на 25% (или на 50% или на 100%) больший исходного. Пусть его длина выбрана равной M, а исходного массива – $N < M$. Некоторое значение (например, -1) будем использовать как маркер «пустой ячейки» в этом втором массиве. Для каждого очередного элемента входного массива вычислим хэш-функцию $h(x) = x \% M$; значение которой мы будем использовать как индекс во втором массиве: и по этому индексу сохраним во второй массив это очередное значение x из исходного массива. Если ячейка занята (то есть, по индексу $I=h(x)$ лежит неотрицательное значение), то заглянем в следующий по порядку элемент второго массива, если он свободен – сохраним значение туда, если нет – сдвинемся еще на одну позицию дальше. Если в таком поиске мы окажемся на последнем элементе второго массива и он занят – то просто «обернемся» в начало: продолжим поиск свободной ячейки начиная с первого элемента. Так как второй массив больше – то свободную ячейку мы найдем довольно быстро. Таким образом мы сохраним во втором массиве (хэш-таблице) все значения исходного массива. Поиск в этой хэш-таблице устроен аналогично: если мы ищем значение y, то вычисляем хэш-функцию от него $h(y)$ и с этим индексом обращаемся в хэш-таблицу. Если ячейка свободна – значит y в массиве нет, если занята – то мы сравниваем y с лежащим в хэш-таблице значением. Совпало – значит мы нашли искомое, не совпало – переходим к следующей ячейке (опять, с «оборотом» через последний элемент, если это необходимо).

Глава 6. Указатели и динамическая память

Что такое указатель?

Указатель – это переменная, содержащая в программе в качестве значения некоторый адрес: поименованной величины или просто ячейки памяти.

В отличие от базовых типов языка Си, указатель — это так называемый производный тип: формируемый из базового при помощи некоторых символов языка, в данном случае применяется символ * (звёздочка). Вот как выглядят простые объявления указателей в программе:

```
int *iPtr; // указатель с именем iPtr на значение типа int
char *cPtr; // указатель с именем cPtr на значение типа char
```

Именами типов в этих примерах являются последовательности *int** и *char**, соответственно. Так как величины могут занимать несколько соседних байт в памяти, а память адресуется побайтно, помимо «звёздочки» в указании типа переменной-указателя присутствует и тип самой величины, на которую "указывает" адрес. Важно также понимать, что пользоваться указателем допустимо лишь после того, как он получит значение в результате инициализации (которая в этих примерах отсутствует!) или значение будет присвоено в процессе работы программы.

Примеры инициализации указателей:

```
int k = 5, m = 3;    // целые переменные, на которые будем создавать указатели
int *kPtr = &k;      // указатель с именем kPtr на значение типа int в переменной k

int **pPtr = &kPtr; // а это – "указатель на указатель", так как kPtr - самостоятельная
                    // переменная типа int*, то у неё есть свой адрес, поэтому можно
                    // создать и указатель, который будет на нее указывать

int *kPtr2 = &k, *mPtr = &m; // тут в одном выражении объявляются два указателя,
                             // обратите внимание, как ставятся "звездочки" при этом;
                             // также подчеркнем: на одну и ту же переменную k
                             // никто не запрещает создать несколько указателей!

void *vPtr = &k;     // создаем указатель "на неизвестный тип" (void), для того, чтобы
                     // указываемым значением воспользоваться, его обязательно нужно
                     // будет явно преобразовать к указателю на какой-то известный тип
```

Здесь мы видим целочисленную переменную *k*, уже имеющую конкретное значение (это важно для последующего), и указатель *kPtr*, инициализируемый адресом этой переменной (символ &, называемый амперсанд, обозначает оператор взятия адреса величины, указываемой после него). Стоит ещё раз отметить, что иметь указатель на неинициализированную величину бесполезно, поскольку используется указатель для доступа к тем величинам, на которые "указывает", позволяя "безымянный" доступ – без указания имени (в примере выше мы можем получить значение переменной k, не зная её имени, а зная лишь её адрес).

Напомним, что синтаксис языка Си позволяет добавлять пробельные символы (табуляции, перевода строки и сами пробелы) в программу, если это не нарушает суть программы. Поэтому формально записать объявление указателя можно по-разному: *int* kPtr*; или *int *kPtr*; или даже так: *int*kPtr*; (хотя, конечно, последнее объявление смотрится странно).

Операции с указателями

Точно так же, как и другие типы величин в языке, указатели допускают осуществление над ними некоторых операций: разыменования, инкремента, декремента и, соответственно, прибавления или вычитания целочисленного значения, а также сравнения указателей. Рассмотрим каждую из них подробнее.

Разыменование

Это наиболее важная операция с указателями, поскольку они, как уже говорилось выше, содержат (при правильном использовании) адреса значений, а необходимо иметь доступ и к самим значениям. Операция разыменования, обозначаемая в языке Си тоже символом «звёздочка», позволяет по указателю (с адресом величины) получить значение самой величины, то есть, то значение, на которое он "указывает". Операция разыменования, в отличие от обозначаемой такой же звездочкой бинарной операции умножения, является унарной: принимающей только один операнд – указатель.

Для предшествующего примера с указателем *kPtr* значением выражения **kPtr* будет целочисленная величина типа *int*, находящаяся по адресу, содержащемуся в указателе. А так как инициализирован указатель адресом переменной k, то **kPtr* даёт текущее значение переменной k, т. е., число 5.

Сравнение объявления указателя и операции разыменования обнаруживает наличие некоторой

двойственности в приводимых выше примерах. С одной стороны, в объявлении указано, что переменная с именем *kPtr* имеет тип *int**, с другой стороны, это и напоминание: если мы будем разыменовывать содержащийся в *kPtr* указатель (то есть, получать величину **kPtr*), то мы получим величину типа int.

Инкремент/декремент

Эти операции изменяют адрес, хранимый указателем, в сторону увеличения или уменьшения адреса. А вот каким будет это изменение – зависит от типа указателя (*Type* *), точнее – от размера в памяти той величины, на которую он указывает (этот размер легко определить, он равен *sizeof*(*Type*)).

В любом случае адрес оказывается изменён так, чтобы указывать на следующую или предыдущую величину этого же типа (предполагается, что величины расположены в памяти "вплотную", без каких-либо "пустых" промежутков). Такое поведение указателя по отношению к этим операциям позволяет последовательно переходить от одной величины в памяти к другой в любом направлении (вперёд или назад), давая возможность их перебирать с какой-либо целью (для вывода, изменения, поиска и т.п.).

Прибавление/вычитание целого

Здесь изменение адреса тоже может быть сделано в сторону увеличения или уменьшения, правда, последняя операция (вместе с операцией декремента) должна использоваться с особой осторожностью - чтобыуказатель не получил недопустимое значение адреса (скажем, выходящее за пределы выделенной памяти).

Заметим, что целое значение, модифицирующее значение указателя, приводит к такому изменению адресав байтах, чтобы новое значение указателя можно было использовать для извлечения/изменения величины соответствующего типа в памяти, т.е., добавление целого значения *k* к указателю p типа *Type* * изменит значение адреса в нём на величину *k*sizeof*(*Type*).

С помощью указателей можно "перемещаться" (в цикле) по последовательности расположенных "вплотную" однотипных величин с помощью **p*++ (вперёд) или **p*-- (назад); значение указателя при этом изменяется (тут надо вспомнить таблицу приоритетов: инкремент более приоритетен, чем разыменование). Можно делать почти то же самое, изменяя смещение *k*при нём (сам указатель остаётся неизменным):

*(*p*+*k*) или *(*p*-*k*);

Массивы и указатели

Важно понимать, что любой массив определяется адресом его самого первого элемента. Поэтому имя массива в операциях с указателями приводится к адресу этого элемента:

```
int A[3] = { 1, 2, 3};
int *p = A; // создаем указатель на первый по порядку элемент массива: &A[0]
```

Операции с массивами можно выполнять как через индексацию с помощью квадратных скобок *x*[*i*], так и через разыменование соответствующего указателя *(*x*+*i*) Более того, компилятор не будет считать ошибкой обращение *i*[*x*], хотя оно и выглядит странновато. Это происходит именно потому, что имя массива компилятор заменяет на его адрес. Соответственно, все три варианта сводятся к обращению через указатель *(*x*+*i*).

Сравнение указателей

Как видно из предыдущего раздела, указатель – это тоже целочисленная величина (смещение от виртуального «начала оперативной памяти» компьютера). Следовательно, значения двух указателей можно сравнивать:

```
if ( kPtr == mPtr ) ...  // ложь, так как они в примере выше указывают на
                         // две разные переменные и у этих переменных точно разные
                         // адреса

int *p2 = &A[1];         // создаем указатель на второй по порядку элемент массива
if ( p2 < &A[2] ) ...    // сравнение имеет смысл, так как указатель и второй
                         // элемент операции сравнения указывают внутрь одного
                         // и того же массива, непрерывного в памяти

if ( p2 < kPtr ) ...     // а эти два сравнения - полная бессмыслица, так как
if ( mPtr >= kPtr ) ...  // p2 указывает на элемент массива, а kPtr - на переменную
                         // к, которая неизвестно, где будет расположена в памяти,
                         // так что результат такого сравнения - непредсказуем!
```

Опасности указателей

Не следует забывать и об "опасностях" работы с указателями. В принципе, указателю можно присвоить любое значение адреса, но вот можно ли обратиться к величине по этому адресу? Это большой вопрос... Поэтому программа с применением указателей весьма чувствительна к ошибкам в их значениях (адресах в памяти) и последствия здесь могут быть самые разные: от неправильного функционирования до прекращения работы программы вслучае её обращения в "запретные" места (об этом обычно "заботится" операционная система, запустившая программу). Так что важно всегда верифицировать значения, присваиваемые указателям и контролировать их возможные изменения, удерживая в допустимых пределах. Приведем пример указателя с некорректным адресом:

```
int *p3 = p2 + 8; // указатель p2 указывает на второй элемент небольшого массива,
                  // а указатель p3 мы проинициализировали адресом, который выходит
                  // за пределы этого массива, если его разыменовать, то программа,
                  // скорее всего, завершится с ошибкой, либо будет получен
                  // некорректный (к тому же случайный) результат ее работы!
```

Нежелательно создавать указатели, которые «никуда не указывают»:

```
int *p4; // указатель p4 не инициализирован, значение адреса в нем будет случайно,
         // а попытка его разыменовать приведет к плохим последствиям, как и в
         // предыдущем примере
```

Вместо этого указатель можно инициализировать специальным «нулевым» адресом:

```
int *p4 = NULL; // такой указатель при попытке разыменования гарантированно
                // приведет к аварийному завершению программы и ошибку найти
                // будет гораздо легче!
```

Функции работы с динамической памятью

Обычная память под переменные программы выделяется компилятором при создании исполняемого файла программы и эта память либо является частью "тела" самой программы (глобальная и статическая память); либо это память под локальные переменные функции, которые располагаются в так называемом стеке программы.

Помимо такой памяти для размещения переменных и массивов величин разного типа, можно также запрашивать память у операционной системы с помощью специальных библиотечных функций. Эта память называется динамической; программа заранее не знает, где будет располагаться эта память и сколько её понадобится, поэтому ничего кроме указателя на начало выделенного блока памяти она получить не может.

Рассмотрим прототипы основных функций для работы с динамической памятью:

```
void* malloc(size_t size);
```

Функция выделяет блок динамической памяти размером *size* (в байтах) и возвращает указатель на его начало (самый первый байт). Если память выделить не удалось (памяти не хватило), возвращается специальное значение указателя: *NULL* (так называемый "невозможный" указатель) и нужно обязательно проверять – что нам вернули?

```
void* realloc(void* ptr, size_t size);
```

Функция изменяет размер ранее выделенного блока динамической памяти с адресом начала *ptr* до размера *size* (в байтах) и возвращает указатель на его начало (самый первый байт). Если память выделить не удалось, возвращается *NULL*. Пользовательское содержимое блока в пределах размера *size* в любом случае остаётся неизменным. Если память выделена, возвращается указатель на её начало, возможно, с другим значением адреса (если выделение произведено в другом месте памяти).

```
void free(void* ptr);
```

Функция объявляет ранее выделенный блок динамической памяти с адресом начала *ptr* "ненужным"; чтобы ошибочно не воспользоваться далее этим адресом, указателю с таким значением разумно изменить значение на *NULL*.

Упомянутые функции выделения динамической памяти никак не инициализируют её; это означает, что в выделенном блоке байты могут иметь произвольное значения (оставшиеся после предыдущих выделений памяти) и не должны использоваться без размещения там нужных программе значений. Возвращаемые и принимаемые значения типа void* символизируют специальный тип "обобщённого" указателя: передать функции с подобным типом параметра можно указатель любого типа, а возвращаемый указатель необходимо всегда преобразовывать к нужному нам типу:

```
int* p = (int*)malloc(size);
```

Организация контейнеров данных

Безадресная память (как явствует из её названия) не предполагает при сохранении указания адреса, где будет находиться величина. Это простой и хороший вариант: не требуется указывать никаких параметров,а лишь само действие размещения где-то в памяти: *push*(), "поместить". Для извлечения величин тоже хотелось бы поступать аналогично, не указывая ничего дополнительно: *pop*(), "вернуть"; величина будет возвращена функцией. Но вот какая из величин в этой памяти должна быть возвращена? Здесь надо принять некоторое разумное соглашение. Конечно, если в такой безадресной памяти сначала не было ничего, то, делая после "заталкивания" величины операцию извлечения, мы ожидаем получить то, что там было. Тут всё понятно. А если что-то в этой безадресной памяти уже находилось?

Имеет смысл хранить все величины последовательно (в противном случае надо было бы ещё запоминать порядок их поступления: после извлечения одной надо понимать, какая будет следующей). Нетрудно сообразить, что

в этой нашей последовательной памяти "выделенными" будут лишь две величины: первая и последняя (из остальных непонятно, что можно выбрать, не указывая больше ничего дополнительно). Таким образом, можно принять, например, такое соглашение: возвращается при извлечении последняя добавленная величина.

Стек

Нас можно поздравить: мы "придумали" безадресную память, называемую стек. Можно запоминать величины, можно их извлекать (ничего "лишнего" указывать не нужно), но извлекается всегда величина, поступившая последней (такова цена за отсутствие дополнительных указаний). Аналогией подобному взаимодействию памяти и значений может служить, например, детская пирамидка с кружками на штыре: мы добавляем туда кружки по одному, складывая их друг на друга, но извлечь оттуда один кружок можно, только если он будет верхним, т.е., последним добавленным. Ещё один пример – стопка книг в узкой коробке (чтобы нельзя было взять книгу из середины стопки, а только сверху).

Но нужна ли такая странная память? Оказывается, да. И весьма часто используется. Например, так осуществляется вызов функций практически в любом языке программирования: перед вызовом функции адрес следующей инструкции помещается в стек, поэтому тогда, когда исполнение функции будет завершено, остаётся только извлечь этот адрес из стека - и выполнение программы можно продолжить.

Именно так работает оператор возврата *return*: из функции как бы предлагается "возвратиться" в место исполнения программного кода, которое следует сразу после вызова. И это очень удобно, поскольку функция может быть вызвана из разных мест программы; возврат автоматически будет туда, откуда она была вызвана. Возможные многочисленные вложенные друг в друга вызовы функций здесь тоже предусмотрены, потому что другие адреса возврата будут далее размещаться в стеке, не нарушая ранее уже занесённые адреса.

Такой вариант взаимодействия с памятью традиционно называется LIFO, это сокращение от выражения Last In First Out, то есть, «последним вошёл, первым вышел»).

Дадим формальное определение стека:

Стек (по-английски «стопка») - это динамическая структура данных (с упорядоченным набором элементов), в которой добавление новых элементов и удаление имеющихся производится с одного конца (он называется вершиной стека).

Таким образом, у стека должны быть как минимум две операции: занесение, функция *push()*, и извлечение, функция *pop*(), но для реальной работы их может понадобиться и больше: удобно, например, знать, что из стека ничего извлечь нельзя (он пуст), иметь возможность проверить, что места для последующего занесения не осталось (стек заполнен); часто полезно проверять величину в вершине стека без её извлечения.

Организовать стек можно из любого упорядоченного хранилища однотипных элементов, например, из одномерного массива, снабдив его параметром, называемым указателем стека, в виде указателя или просто индекса. Указатель или индекс могут быть связаны как с последним размещённым в стеке элементом, так и, как вариант, с первым "свободным" местом после только что добавленного элемента.

Если места в хранилище для очередного размещения не будет хватать (всё свободное уже было заполнено), память можно перевыделить с помощью функции *realloc()*, чтобы её стало больше, причём с определённым запасом; в зависимости от выбираемого способа увеличения размера памяти различают стратегии удвоения (размер памяти увеличивается вдвое) или изменения на константу (размер памяти увеличивается на некоторую постоянную величину).

И еще нужно решить техническую проблему: если размер выделенной памяти будет изменен, то и адрес, который на эту память указывает, может измениться, его нужно будет сообщить «вызывающей стороне». Для этого удобно сделать типом данных, который будет обозначать текущий стек, тип «указатель на указатель»: мы передаем в функции адрес указателя на стек и, если он внутри функции будет изменен, то вызывающая сторона получит это изменение. Аналогичный прием можно использовать и для возврата значений предельной емкости стека и количества реальных элементов в этом стеке.

В результате программный интерфейс стека может выглядеть так:

- добавить элемент в стек:

```
void push(double **stack, size_t* size, size_t* capacity, double element);
```

- получить элемент из стека:

```
double top(double **stack, size_t size);
```

- удалить элемент из стека:

```
void pop(double **stack, size_t* size);
```

- очистка памяти:

```
void destroy(double **stack);
```

При помещении первого элемента в новый стек можно в качестве адреса стека передавать просто нулевой указатель:

```
double* pStack = NULL;
size_t size = 0, capacity = 0;
push(&pStack, &size, &capacity, 6.02E+23);
```

При этом функция *push()* самостоятельно должна выделить некоторое начальное количество динамической памяти под стек и заполнить все возвращаемые по указателям значения. Аналогично, разумным будет, если функция *destroy()* будет обнулять указатель на уничтоженный стек. Отметим еще один нюанс предлагаемой реализации: функция удаления элемента из стека не очищает память, таким образом, выделенная динамическая память при любой работе со стеком – только растет, а освобождается она только один раз – при вызове функции *destroy()*.

Очередь

А что, если в упомянутую выше безадресную память заносить величины, а извлекать их потом не "с того же конца", а из начала? Тогда получится безадресная память с организацией FIFO (сокращение выражения First In First Out, «первым вошёл, первым вышел»): это очередь.

Подлежащие обслуживанию люди (события, предметы) формируют очередь (упорядоченную последовательность), добавляясь в её конец, а обслуживаются, начиная с начала очереди. Здесь аналогия с обычной людской очередью более чем очевидна, а потому дополнительного объяснения не требует.

Формальное определение очереди:

Очередь - это динамическая структура данных (с упорядоченным набором элементов), в которой добавление новых элементов производится с одного конца ("хвост"), а извлечение имеющихся - с другого ("голова").

Таким образом, для очереди должны быть (как минимум) определены операции: добавление элемента в конец и извлечение элемента из начала. Удобно иметь способ проверки очереди на "пустоту" (когда извлечь ничего нельзя); желательна также проверка на невозможность добавления: если очередь организуется в ограниченной области памяти (например, статической), это позволит "не переполнить" её, если используется динамическая память - вовремя увеличить её размер.

Реализовать очередь можно с помощью массива, в котором элементы извлекаются из начала массива, сдвигая оставшиеся элементы к началу и добавляются в конец (возможны и другие варианты исполнения).

Программный интерфейс очереди при этом может выглядеть в точности так же, как выглядел интерфейс стека:

- добавить элемент в очередь:

```
void push(double** queue, size_t* size, size_t* capacity, double element);
```

- получить элемент из очереди:

```
double top(double** queue, size_t size);
```

- удалить элемент из очереди:

```
void pop(double** queue, size_t* size);
```

- очистка памяти:

```
void destroy(double** queue);
```

Но результат работы с очередью будет другим, не таким, как у стека. Это будет определяться реализацией функций, приведенных выше. Работа при первом добавлении элемента в очередь, при удалении элемента и при полной очистке очереди аналогична тому, как это было описано в разделе про стек.

Дека

Дека - это обобщение очереди. Англоязычное название deque образовано как полу-аббревиатура от английского Double Ended Queue, очередь с двумя концами. В деке быстро добавлять и извлекать элементы можно с любого конца.

Дека - это динамическая структура данных (с упорядоченным набором элементов), в которой добавление новых и удаление имеющихся элементов может производиться с обоих её концов.

Таким образом, для деки должны быть определены операции: добавление элемента в начало, добавление элемента в конец, извлечение элемента из начала, извлечение элемента из конца. При реализации нужно не забыть сделать проверку деки на "пустоту" (когда извлечь ничего нельзя ни с одной стороны); желательна также проверка на невозможность добавления элементов с каждой из сторон: если дека организуется в ограниченной области памяти, это позволит "не переполнить" её, если же используется динамическая память, то можно вовремя увеличить размер памяти под деку.

Реализовать деку можно с помощью массива, в котором элементы начинают размещаться "в центре" и добавляются по обе стороны от этого центра (возможны и другие варианты исполнения).

- добавить элемент в начало деки:

```
void push_front(double** deque, size_t* begin, size_t* end,
                size_t* capacity, double element);
```

- добавить элемент в конец деки:

```
void push_back(double** deque, size_t* begin, size_t* end,
                size_t* capacity, double element);
```

- получить элемент из начала деки:

```
double get_front(double** deque, size_t begin);
```

- получить элемент с конца деки:

```
double get_back(double** deque, size_t end);
```

- удалить элемент из начала деки:

```
void pop_front(double** deque, size_t* begin, size_t end);
```

- удалить элемент с конца деки:

```
void pop_back(double** deque, size_t begin, size_t* end);
```

- очистка памяти:

```
void destroy(double** deque);
```

Обратите внимание: при удалении элементов с разных концов деки по-разному передаются параметры начала и конца деки: из них только один из параметров будет меняться, поэтому только один из них передается по указателю, а второй – по значению.

Работа при первом добавлении элемента в деку, при удалении элемента и при полной очистке деки аналогична тому, как это было описано в разделе про стек.

Обратите внимание, что из деки можно сделать и обычную очередь и стек: можно просто не использовать часть возможных для деки операций.

Вектор (динамический массив)

Вектор — это динамический массив, который сам следит за своим размером. Все детали реализации функций описаны выше, например, в разделе про стек, тут приведем только программный интерфейс вектора:

- вставка элемента в позицию **index**:

```
void insert(double** array, size_t* size, size_t* capacity,
            size_t index, double element);
```

- получение элемента на позиции **index**:

```
double at(double** array, size_t index);
```

- изменение размера вектора:

```
void resize(double** array, size_t* size, size_t* capacity, size_t new_size);
// подчеркнем, что тут можно и увеличивать размер вектора, инициализируя новые
// элементы, например, нулями, и уменьшать его: при этом обязательно должны
// сохраниться значения остающихся элементов!
```

- очистка памяти:

```
void destroy(double** array);
```

Двумерные динамические массивы

В программах на языке Си элементы статического двумерного массива (матрицы) могут храниться в памяти построчно (без промежутков), строка за строкой: сначала элементы первой строки, потом - элементы следующей строки и так далее. Это позволяет вычислить, где начинается любая строка: первая (с индексом 0) располагается в самом начале, вторая начинается через W элементов (где W - размер строки матрицы в элементах), третья - через 2*W (после двух строк) и т.д. Поэтому адрес начала строки матрицы с индексом i при таком её расположении даётся выражением p+i*W, где p – адрес начала матрицы, а адрес элемента с индексами i, j - выражением p+i*W+j. Сам элемент может быть получен после операции разыменования: *(p+i*W+j). Напомним, что это выражение полностью эквивалентно такому: p[i*W+j].

При вычислении выражения типа A[i][j] первая операция индексации должна давать указатель на начало *i*-й строки матрицы. Для этого достаточно выделить дополнительно вспомогательный (одномерный) массив указателей с числом элементов, равным количеству строк в матрице.

Где при этом будут располагаться сами строки? Возможны как минимум два варианта: либо каждая строка – это отдельно выделяемый фрагмент динамической памяти (каждый – со своим адресом начала строки), либо можно выделить один фрагмент на всё содержимое матрицы, а массив указателей заполнить адресами начала каждой строки (либо вообще не использовать массив указателей на начала строк матрицы, тогда придется

сделать один метод «дай i,j элемент»).

Используем первый способ и выделим память под указатели на адреса начала каждой из строк матрицы:

```
double** a=(double**)malloc(n*sizeof(double*));
```

После этого нужно выделим память для каждой строки матрицы:

```
for(int i=0; i<n; i++)
	a[i]=(double*)malloc(m*sizeof(double));
```

При удалении такого массива придется сначала удалить строки:

```
for(int i=0; i<n; i++)
	free(a[i]);
```

А потом удалить указатели на них:

```
free(a);
```

Оформим эти действия в виде функций

```
double** create_matrix(int n, int m) // создание массива n*m
{
	double** a=(double**)malloc(n*sizeof(double*));
	for(int i=0; i<n; i++)
		a[i]=(double*)malloc(m*sizeof(double));
	return a; // возвращаем созданный массив
}
void delete_matrix(double** a, int n)// освобождение массива из n строк
{
	for(int i=0;i<n;i++)
		free(a[i]);
	free(a);
}
```

Добавим теперь функции ввода-вывода

```
void print_matrix(double** a, int n, int m)// вывод матрицы на экран
{
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<m;j++)
		{
			printf("%lf\t",a[i][j]);
		}
		printf("\n");
	}
	printf("\n");
}
void input_matrix(double** a, int n, int m)// ввод матрицы с клавиатуры
{
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<m;j++)
		{
			printf("[%d][%d]:",i,j);
			scanf("%lf",&a[i][j]);
		}
	}
	getchar(); // убрать оставшийся от ввода символ '\n'
}
```

При этом получение i,j-элемента матрицы можно реализовать обычным для языка Си образом:

```
double x = a[i][j];
```

Но можно и сделать отдельную функцию для этого:

```
double matrix_element(double** a, int i, int j) // вернуть i,j элемент матрицы
{
	return a[i][j];
}
```

Обработку данных тоже целесообразно оформлять в виде функций. Рассмотрим пример обработки суммирующий значения столбцов матрицы и заполняющий заранее выделенный под эти суммы массив:

```
// функция суммирующая столбцы матрицы a[n][m] в массив res[m]
void sum(double** a, int n, int m, double* res)
{
	for(int j=0;j<m;j++)
	{
```

```
            res[j]=0;
            for(int i=0;i<n;i++)
            {
                  res[j]+=a[i][j];
            }
      }
}
```

Таким образом, для решения задач по обработке матриц получается полезный набор функций (его можно расширить), который целесообразно записать в отдельный файл и подключить его к проекту.
Для вызова этих функций необходимы объявления (прототипы) этих функций, которые можно поместить в заголовочный файл (например array.h)

```
double** create_matrix(int n, int m);             // создание матрицы n*m
double matrix_element(double** a, int i, int j); // вернуть i,j элемент матрицы
void delete_matrix(double** a, int n);            // освобождение матрицы из n строк
void print_matrix(double** a, int n, int m);      // вывод матрицы на экран
void input_matrix(double** a, int n, int m);      // ввод матрицы с клавиатуры
void sum(double** a, int n, int m, double* res); // суммирование столбцов матрицы
                                                  // в массив res[m]
```

После этого решение задания на матрицы практически сведётся к последовательному вызову функций:

```
#include <stdio.h>
#include <stdlib.h>
#include "array.h"

int main()
{
      int n=3, m=4; // задали размер массива
      double** a=create_matrix(n, m);// создали массив
      input_matrix(a, n, m); // ввели
      print_matrix(a, n, m); // вывели
      double* s=(double*)malloc(m*sizeof(double)); // создали массив результата
      sum(a,n,m,s); // обработали массив и получили результат
      for(int i=0; i<m; i++)
            printf("%lf\t", s[i]); // вывели результат
      delete_matrix(a, n); // освободили исходный массив
      free(s);   // освободили массив результата
      getchar(); // задержка закрытия окна
      return 0;
}
```

Варианты задач для решения

В типовой задаче необходимо написать реализацию требуемого контейнера и тестовую функцию *main*() для нее, которая будет выполнять описанные в задаче действия и тестировать все реализованные для контейнера функции. Если стратегия увеличения ёмкости контейнера не указана – реализовать стратегию удвоения памяти при её нехватке. Если не указан тип элемента контейнера – реализовать для типа *double*. Контейнер заполнять вводимыми с консоли элементами.

Реализацию функций контейнера нужно поместить в отдельный файл (например, *array.cpp*), функцию тестирования поместить в другой файл (например, *task6.cpp*). Объявления функций поместить в заголовочный файл (например, *array.h*).

1. Вариант Реализовать отсортированный вектор для элементов типа **double** со стратегией удвоения ёмкости, который при вставке новых элементов сохраняет свою сортировку.
2. Вариант Реализовать сортировку вектора вещественных чисел без его изменения: создается дополнительный динамический массив указателей на элементы исходного массива. Дальше производится сортировка этого массива указателей, без изменения исходного вектора чисел.

3. Вариант

Есть два динамических массива: массив «ключей» и массив «значений». Также есть третий динамический массив, в котором попарно (один за другим) хранятся указатели на соответствующие элементы первых двух массивов. Необходимо реализовать функцию сортировки пар в третьем массиве по возрастанию ключей, а также функцию добавления новой пары ключ-значение ко всей коллекции (с изменением всех трех массивов).

4. Вариант

Есть два динамических массива: массив «ключей» и массив «значений». Также есть третий массив, в котором попарно (один за другим) хранятся указатели на соответствующие элементы первых двух массивов. Необходимо реализовать функцию удаления пар ключ-значение с дубликатами ключей: для двух пар с одинаковыми ключами, оставляется та пара, у которой значение больше.

5. Вариант

Реализовать стек значений со стратегией удвоения ёмкости памяти при её нехватке.

6. Вариант

Реализовать стек значений со стратегией увеличения ёмкости памяти на константу при её нехватке. (константа задается через #define)

7. Вариант

Реализовать очередь значений, используя в качестве указателя начала очереди индекс первого элемента в очереди, а в качестве указателя конца очереди индекс первого свободного элемента. Использовать стратегию удвоения ёмкости памяти при её нехватке.

8. Вариант

Реализовать очередь значений, используя в качестве указателя начала очереди идекс первого элемента в очереди, а в качестве указателя конца очереди индекс последнего размещенного элемента. Использовать стратегию увеличения ёмкости памяти на константу при её нехватке. (константа задается через #define)

9. Вариант

Реализовать деку значений, используя в качестве указателя начала деки индекс первого элемента, а в качестве указателя конца деки индекс последнего размещенного элемента.

10. Вариант

Реализовать деку значений, используя в качестве указателя начала деки индекс первого элемента, а в качестве указателя конца деки индекс первого свободного элемента.

11. Вариант

Реализовать двумерную матрицу, для которой реализованы функции взятия элемента по двум индексам (номер строки и номер столбца) и возможностью вставить как новый столбец, так и новую строку. Матрицу реализовать как массив указателей на массивы, хранящие ее строки.

12. Вариант

Реализовать двумерную матрицу, для которой реализованы функции взятия элемента по двум индексам (номер строки и номер столбца) и возможностью вставить как новый столбец, так и новую строку. Матрицу реализовать посредством одного динамического массива.

13. Вариант

Реализовать двумерную матрицу, для которой реализованы функции взятия элемента по двум индексам (номер строки и номер столбца) и возможностью удалить как указанный столбец, так и указанную строку. Матрицу реализовать как массив указателей на массивы, хранящие ее строки.

14. Вариант

Реализовать двумерную матрицу, для которой реализованы функции взятия элемента по двум индексам (номер строки и номер столбца) и возможностью удалить как указанный столбец, так и указанную строку. Матрицу реализовать посредством одного динамического массива.

15. Вариант

Реализовать функцию, которая перемножает две матрицы и возвращает результат в создаваемой третьей матрице. Матрицы реализовать как массивы указателей на массивы, хранящие их строки.

16. Вариант

Реализовать четыре функции, представляющие собой четыре арифметических действия с другими неизвестными функциями, представленными указателями, а также функцию, реализующую суперпозицию двух неизвестных функций, представленных указателями. В качестве элементарных функций использовать: **sin()**, **cos()**, **log()**, **exp()**, **sqrt(), pow()**. По запросу пользователя – построить сложную функцию, реализующую запрошенное функциональное выражение и посчитать значение получившейся функции в заданной точке.

17. Вариант

Дан целочисленный вектор. Реализовать функцию, которая строит и возвращает в новом векторе гистограмму уникальных значений из него. Количество ячеек гистограммы задается пользователем. Распечатать получившуюся гистограмму в «графическом» виде построчно: в каждой строке терминала вывести необходимое для очередной ячейки гистограммы количество символов «*».

18. Вариант

Реализовать очередь на основе двух динамических массивов. Реализовать функции добавления в конец, добавления в начало и взятия элемента по индексу из этой пары массивов. Будем считать, что первый из двух массивов хранит начало очереди в обратном порядке – то есть, последний элемент массива является головным элементом очереди, предпоследний – вторым и т.д. Элементом очереди, следующим за нулевым элементом первого массива является нулевой элемент второго массива, следом – первый элемент второго массива и т.д. до последнего элемента, второго массива, который является последним элементом очереди. Каждый из массивов содержит заранее выделенный запас элементов, который позволит быстро добавлять и удалять элементы в начало и в конец очереди. Нужно учесть, что когда один из двух массивов опустеет – очередь нужно будет «сбалансировать»: переместить в него половину элементов из второго массива.

19. Вариант

Работа с разреженными матрицами. Дана двумерная вещественная матрица, многие элементы которой являются нулями. Написать функцию для преобразования такой матрицы в упакованный набор: массив значений, массив указателей, переводящий двумерные индексы к позиции в массиве значений, при этом элементы матрицы, по модулю меньшие **10-10**, не сохраняются в упакованный набор. Также написать функцию, которая возвращает **[i][j]**-элемент матрицы и функцию печати такой матрицы в обычном двумерном виде. При заполнении матрицы воспользоваться генератором случайных чисел – сделать так, чтобы примерно каждый третий элемент матрицы был равен нулю.

20. Вариант

Реализовать работу с квадратными верхними треугольными матрицами: создание, удаление, умножение, взятие **[i][j]**-элемента матрицы. Нулевые элементы из нижней треугольной области не хранить.

21. Вариант

Реализовать работу с ленточными квадратными матрицами: создание, удаление, умножение, взятие **[i][j]**-элемента матрицы. Нулевые элементы из верхней и нижней треугольных областей не хранить. Ширина «ленты» ненулевых значений около главной диагонали задается пользователем.

22. Вариант

Реализовать сортированную очередь из строк символов (каждая строка – это динамический массив с элементами типа char). Пользователь вводит текстовые строки, которые добавляются в массив указателей, отсортированный по алфавиту. Сначала пользователь добавляет пять строк подряд, затем после каждой добавляемой строки печатается и удаляется одна строка из головы очереди. Программа завершается после ввода строки «**end**».

23. Вариант

Сортировка слиянием. Дан массив, длина которого равна степени числа 2. Разделить массив пополам на два новых массива. Слить эти два массива в исходный массив, соблюдая порядок сортировки в парах «элемент из первого массива»-«элемент из второго массива». Получившийся массив опять разделить пополам и слить два массива в исходный, соблюдая порядок сортировки для четверок элементов. Далее повторить операции для восьмерок элементов и т.д., пока весь массив не окажется отсортированным. В конце – удалить служебные массивы.

24. Вариант

Реализовать двумерную матрицу произвольной размерности, хранящую отдельные биты (так называемый bitmap) в виде битового вектора. Биты хранить упакованными по восемь бит – в байтах. Реализовать функцию, которая возвращает **[i][j]**-бит из матрицы в виде булевского значения. Реализовать функции вставки новых строки и колонки бит.

25. Вариант

Найти максимальный элемент каждой строки матрицы и записать их в стек. Затем извлечь все значения из стека и распечатать их.

26. Вариант

Написать программу, которая удаляет все близкие к указанному пользователем значению элементы вещественного динамического массива. Допустимая степень близости значений задается при помощи #define.

27. Вариант

Построить вектор из порядковых номеров максимальных по модулю элементов в столбцах матрицы.

28. Вариант

Для данной матрицы построить два динамических массива: со средним значением каждого столбца и со стандартным уклонением величин в нём.

29. Вариант

Построить вектор, в котором попарно (один за другим) будут сохранены индексы i и j элементов матрицы, максимальных по модулю в каждом её столбце.

30. Вариант

Создать матрицу из копий полных строк целочисленной исходной матрицы, которые содержат либо минимальный, либо максимальный элемент во всей исходной матрицы. Учтите, что и минимальный и максимальный элементы могут повторяться в разных строках. Обе матрицы реализуйте в форме единых динамических массивов их элементов.

Глава 7. Работа с файлами, текстовый ввод и вывод.

При работе с файлами можно воспользоваться функциями форматированного ввода/вывода, если необходимо записывать в файл или читать из файла, например, результаты расчётов.

Пусть, например, нам требуется протабулировать функцию $y(x) = ax^2 + bx + c$ в диапазоне от x_0 до x_N и сохранить результаты в файл. Для этого напишем программу:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

/* Функция для расчёта полинома */
double poly(double a, double b, double c, double x) {
	return a * x * x + b * x + c;
}

int main(void) {
	double a, b, c, x0, xN, dx, x, y;
	int N, n;
	FILE *f;
	char fn[] = "a.txt";

	printf("Enter coefficients a, b, c: ");
	/* Вводим коэффициенты полинома */
	scanf(" %lf%lf%lf", &a, &b, &c);
	printf("Enter x0 and xN: ");
	/* Вводим пределы */
	scanf(" %lf%lf", &x0, &xN);
	printf("Enter n: ");
	/* Вводим количество интервалов */
	scanf(" %d", &N);
	dx = (xN - x0) / N;
	x = x0;
	/* Открываем текстовый файл на запись */
	f = fopen(fn, "w");
	/* Проверка открытия файла */
	if(f == NULL) {
		printf("Cannot open file \"%s\" for output.\n", fn);
		return 2;
	}
	for(n = 0; n <= N; n++) {
		y = poly(a, b, c, x);
		/* Выводим таблицу значений в файл */
		fprintf(f, "%3d %-10.2f%+12.4e\n", n, x, y);
		x += dx;
	}
	/* Закрываем файл */
	fclose(f);
	/* Открываем текстовый файл на чтение */
	f = fopen(fn, "r");
	/* Проверка открытия файла */
	if(f == NULL) {
		printf("Cannot open file \"%s\" for input.\n", fn);
		return 2;
	}
	/* Читаем данные из файла и выводим их на экран */
	printf("  n       x         y\n");
	while(fscanf(f, " %d%lf%le", &n, &x, &y) != EOF) {
		printf("%3d%8.3f%8.3f\n", n, x, y);
	}
	/* Закрываем файл */
	fclose(f);
	return 0;
}
```

Первый вызов функции *scanf*() вводит коэффициенты полинома. Первый символ форматирующей строки — пробел заставляет функцию пропустить все пробельные символы в потоке ввода. Второй и третий вызовы *scanf*() работают аналогично. Рассмотрим подробно структуру форматирующей строки функции *fprintf*(). Первые три символа (%3d) означают вывод целого числа в поле шириной три символа с правым выравниванием (спецификатор d — целое, флаг 3 — ширина поля вывода). Далее идёт символ пробел — в поток вывода будет выведен символ пробела. Следующие символы (%-10.2f) означают вывод величины с плавающей точкой с двумя знаками после десятичного разделителя в поле шириной 10 символов с левым выравниванием (спецификатор f — переменная типа double или float, флаг «минус» — выводимая величина выравнивается по левому краю, 10 — ширина поля вывода, .2 — количество знаков после десятичного разделителя). Идущая далее группа символов (%+12.4e) означает вывод величины с плавающей точкой в экспоненциальном формате с 4 знаками после десятичного разделителя в поле шириной 12 символов с правым выравниванием, перед числом ставится его знак, даже если значение положительное (спецификатор e — переменная типа *double* или *float*, выводится в экспоненциальном виде, флаг «плюс» — вывод знака числа, 12 — ширина поля вывода, .4 — количество знаков после десятичного разделителя). Следующие далее символы (\n) означают переход на новую строку. Результат работы этой программы (ввод пользователя выделен полужирным шрифтом):

```
Enter coefficients a, b, c: -1.5 2 3
Enter x0 and xN: 2 3
Enter n: 10
  n      x       y
  0   2.000   1.000
  1   2.100   0.585
  2   2.200   0.140
  3   2.300  -0.335
  4   2.400  -0.840
  5   2.500  -1.375
  6   2.600  -1.940
  7   2.700  -2.535
  8   2.800  -3.160
  9   2.900  -3.815
 10   3.000  -4.500
```

Содержимое файла:

```
  0 2.00       +1.0000e+00
  1 2.10       +5.8500e-01
  2 2.20       +1.4000e-01
  3 2.30       -3.3500e-01
  4 2.40       -8.4000e-01
  5 2.50       -1.3750e+00
  6 2.60       -1.9400e+00
  7 2.70       -2.5350e+00
  8 2.80       -3.1600e+00
  9 2.90       -3.8150e+00
 10 3.00       -4.5000e+00
```

Варианты задач для решения

1. Вариант Напишите программу, которая запрашивает у пользователя целое число `N` и вещественное число `x` и записывает в текстовый файл таблицу значений функции $y_n(x) = sin(nx)cos^2(x), n = \overline{1,N}$ в две колонки (в левой *n*, в правой y_n, левая колонка имеет ширину 3 символа с выравниванием по правой границе, правая — ширину 8 символов с выравниванием по правой границе, значение y_n выводится с 4 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает его содержимое в другой файл, меняя порядок следования чисел в строках на обратный, при этом левая колонка должна иметь ширину 10 символов с выравниванием по правой границе, значение y_n выводится с 2 знаками после десятичного разделителя, правая колонка выравнивается по левой границе.

2. Вариант Напишите программу, которая запрашивает у пользователя целое число `N` и вещественное число `x` и записывает в текстовый файл таблицу значений функции $y_n(x) = sin(nx)cos(n|x|), n = \overline{1,N}$ в две колонки (в левой *n*, в правой y_n, левая колонка имеет ширину 4 символа с выравниванием по правой границе, правая — ширину 10 символов с выравниванием по правой границе, значение y_n выводится с 6 знаками после десятичного разделителя, перед числом должен стоять его знак, даже если число положительное), закрывает файл. Затем открывает этот же файл на чтение и переписывает его содержимое в другой файл, меняя порядок следования чисел в строках на обратный, при этом левая колонка должна иметь ширину 8 символов с выравниванием по правой границе, значение y_n выводится с 2 знаками после десятичного разделителя, правая колонка должна иметь ширину 3 символа с выравниванием по правой границе.

3. Вариант Напишите программу, которая запрашивает у пользователя два целых числа `N` и `M` и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = \sqrt{|cos(mx_n)| - sin^2(mx_n)}, n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (`N` строк, `M` колонок, колонки имеют ширину 8 символов с выравниванием по правой границе, значение функции выводится с 3 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает его содержимое в другой файл, меняя порядок следования чисел в строках на обратный, при этом колонки должны иметь ширину 10 символов с выравниванием по правой границе, значение выводится с 2 знаками после десятичного разделителя.

4. Вариант Напишите программу, которая запрашивает у пользователя два целых числа `N` и `M` и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = ln(|mx_n|)cos(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (`N` строк, `M` колонок, колонки имеют ширину 12 символов с выравниванием по правой границе, значение функции выводится с 6 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает его содержимое в другой файл, меняя порядок следования чисел в нечётных строках на обратный, при этом колонки должны иметь ширину 7 символов с выравниванием по правой границе, значение выводится с 2 знаками после десятичного разделителя.

5. Вариант Напишите программу, которая запрашивает у пользователя два целых числа `N` и `M` и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = ln(|mx_n|)cos(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (`N` строк, `M` колонок, колонки имеют ширину 8 символов с выравниванием по правой границе, значение функции выводится с 2 знаками после десятичного разделителя, перед числом должен стоять его знак, даже если число положительное), закрывает файл. Затем открывает этот же файл на чтение и переписывает его содержимое в другой файл, меняя порядок следования чисел в чётных строках на обратный, при этом колонки должны иметь ширину 7 символов с выравниванием по правой границе, значение выводится с 3 знаками после десятичного разделителя.

6. Вариант Напишите программу, которая запрашивает у пользователя два целых числа `N` и `M` и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = \sqrt{|x_n|/2 + mx_n^2 sin^2\,(mx_n)/mx_n}\,, n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (`N` строк, `M` колонок, колонки имеют ширину 9 символов с выравниванием по правой границе, значение функции выводится с 5 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только те числа каждой строки, которые превышают среднее арифметическое значение чисел в ней, при этом колонки должны иметь ширину 12 символов с выравниванием по правой границе, значение выводится с 5 знаками после десятичного разделителя.

7. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = msin(mx_n)cos^2(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 8 символов с выравниванием по правой границе, значение функции выводится с 3 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только те числа каждой строки, которые превышают среднее арифметическое значение чисел в ней, меняя порядок их следования на обратный, при этом колонки должны иметь ширину 10 символов с выравниванием по правой границе, значение выводится с 2 знаками после десятичного разделителя.

8. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = \sqrt{|x_n|}sin(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, если дробная часть числа оказывается меньше запрошенного у пользователя значения, оно округляется в меньшую сторону и записывается как целое, колонки имеют ширину 9 символов с выравниванием по правой границе, значение функции выводится с 3 знаками после десятичного разделителя для вещественных чисел и 9 символов для целых чисел), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только целые числа каждой строки, при этом колонки должны иметь ширину 5 символов с выравниванием по правой границе.

9. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = cos(mx_n)ln(|x_n|), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, если дробная часть числа оказывается меньше запрошенного у пользователя значения, оно округляется в меньшую сторону и записывается как целое, колонки имеют ширину 9 символов с выравниванием по правой границе, значение функции выводится с 3 знаками после десятичного разделителя для вещественных чисел и 9 символов для целых чисел), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только целые числа каждой строки, меняя порядок их следования на обратный, при этом колонки должны иметь ширину 5 символов с выравниванием по левой границе, перед числом должен стоять его знак, даже если число положительное.

10. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = exp(-x_n/m)sin(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, если дробная часть числа оказывается меньше запрошенного у пользователя значения, оно округляется в меньшую сторону и записывается как целое, колонки имеют ширину 9 символов с выравниванием по правой границе, значение функции выводится с 3 знаками после десятичного разделителя для вещественных чисел и 9 символов для целых чисел), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только вещественные числа каждой строки, при этом колонки должны иметь ширину 12 символов с выравниванием по правой границе, значение выводится с 4 знаками после десятичного разделителя.

11. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = exp(-x_n^2/m)cos(m^2x_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, если дробная часть числа оказывается меньше запрошенного у пользователя значения, оно округляется в меньшую сторону и записывается как целое, колонки имеют ширину 9 символов с выравниванием по правой границе, значение функции выводится с 3 знаками после десятичного разделителя для вещественных чисел и 9 символов для целых чисел), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только вещественные числа каждой строки, меняя порядок их следования на обратный, при этом колонки должны иметь ширину 10 символов с выравниванием по правой границе, значение выводится с 2 знаками после десятичного разделителя.

12. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = ln(|mx_n|)sin(mx_n^2), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 11 символов с выравниванием по правой границе, значение функции выводится с 5 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только те числа каждой строки, дробная часть которых не превышает заданного значения, запрошенного у пользователя, при этом колонки должны иметь ширину 8 символов с выравниванием по правой границе, значение выводится с 2 знаками после десятичного разделителя.

13. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = cos^2(x_n/m)sin(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 10 символов с выравниванием по правой границе, значение функции выводится с 4 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только те числа каждой строки, дробная часть которых не превышает заданного значения, запрошенного у пользователя, меняя порядок их следования на обратный, при этом колонки должны иметь ширину 9 символов с выравниванием по правой границе, значение выводится с 3 знаками после десятичного разделителя.

14. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = exp(-x_n/m)cos(m|x_n|), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 10 символов с выравниванием по правой границе, значение функции выводится с 5 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только те числа каждой строки, дробная часть которых не превышает заданного значения, запрошенного у пользователя, меняя порядок следования чисел в нечётных строках на обратный, при этом колонки должны иметь ширину 9 символов с выравниванием по правой границе, значение выводится с 3 знаками после десятичного разделителя.

15. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = sin^2(m|x_n|)cos(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 10 символов с выравниванием по правой границе, значение функции выводится с 5 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только те числа каждой строки, дробная часть которых не превышает заданного значения, запрошенного у пользователя, меняя порядок следования чисел в чётных строках на обратный, при этом колонки должны иметь ширину 9 символов с выравниванием по правой границе, значение выводится с 3 знаками после десятичного разделителя.

16. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = sin^2(mx_n)cos(mx_n^2), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 9 символов с выравниванием по правой границе, значение функции выводится с 5 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только те числа каждой строки, которые превышают среднее арифметическое значение чисел в ней, меняя порядок следования чисел в нечётных строках на обратный, при этом колонки должны иметь ширину 12 символов с выравниванием по правой границе, значение выводится с 5 знаками после десятичного разделителя.

17. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = cos(mx_n^2)ln(|x_n|), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, если дробная часть числа оказывается меньше запрошенного у пользователя значения, оно округляется в меньшую сторону и записывается как целое, колонки имеют ширину 9 символов с выравниванием по правой границе, значение функции выводится с 3 знаками после десятичного разделителя для вещественных чисел и 9 символов для целых чисел), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только целые числа каждой строки, меняя порядок следования чисел в нечётных строках на обратный, при этом колонки должны иметь ширину 5 символов с выравниванием по левой границе, перед числом должен стоять его знак, даже если число положительное.

18. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = sin(mx_n)cos(x_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 12 символов с выравниванием по правой границе, значение функции выводится с 6 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только отрицательные числа каждой строки, при этом колонки должны иметь ширину 11 символов с выравниванием по правой границе, значение выводится с 3 знаками после десятичного разделителя.

19. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = sin(mx_n)cos^2(x_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 11 символов с выравниванием по правой границе, значение функции выводится с 4 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только отрицательные числа каждой строки, меняя порядок их следования на обратный, при этом колонки должны иметь ширину 8 символов с выравниванием по правой границе, значение выводится с 2 знаками после десятичного разделителя.

20. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = sin^2(mx_n)cos(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 10 символов с выравниванием по правой границе, значение функции выводится с 3 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только отрицательные числа каждой строки, меняя порядок следования чисел в нечётных строках на обратный, при этом колонки должны иметь ширину 8 символов с выравниванием по правой границе, значение выводится с 2 знаками после десятичного разделителя.

21. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = sin(mx_n)cos^2(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 11 символов с выравниванием по правой границе, значение функции выводится с 5 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только положительные числа каждой строки, удваивая их, при этом колонки должны иметь ширину 8 символов с выравниванием по правой границе, значение выводится с 3 знаками после десятичного разделителя.

22. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = sin(mx_n^2)cos(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 10 символов с выравниванием по правой границе, значение функции выводится с 3 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только положительные числа каждой строки, удваивая их и меняя порядок их следования на обратный, при этом колонки должны иметь ширину 7 символов с выравниванием по правой границе, значение выводится с 2 знаками после десятичного разделителя.

23. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = sin(mx_n^2)cos^2(x_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 11 символов с выравниванием по правой границе, значение функции выводится с 5 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл только положительные числа каждой строки, удваивая их и меняя порядок следования чисел в нечётных строках на обратный, при этом колонки должны иметь ширину 8 символов с выравниванием по правой границе, значение выводится с 3 знаками после десятичного разделителя.

24. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = exp(-x_n/m)sin(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 10 символов с выравниванием по правой границе, значение функции выводится с 4 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл сначала отрицательные, затем положительные числа каждой строки, при этом колонки должны иметь ширину 8 символов с выравниванием по правой границе, значение выводится с 3 знаками после десятичного разделителя.

25. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = exp(-x_n/m)sin(mx_n^2), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 12 символов с выравниванием по правой границе, значение функции выводится с 6 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл сначала удвоенные отрицательные, затем утроенные положительные числа каждой строки, при этом колонки должны иметь ширину 9 символов с выравниванием по правой границе, значение выводится с 4 знаками после десятичного разделителя.

26. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = exp(-x_n/m)cos(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 11 символов с выравниванием по правой границе, значение функции выводится с 4 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл сначала переставленные в обратном порядке отрицательные, затем положительные числа каждой строки, при этом колонки должны иметь ширину 8 символов с выравниванием по правой границе, значение выводится с 2 знаками после десятичного разделителя.

27. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = exp(-x_n/m)cos(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 10 символов с выравниванием по правой границе, значение функции выводится с 3 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл сначала отрицательные, затем переставленные в обратном порядке положительные числа каждой строки, при этом колонки должны иметь ширину 8 символов с выравниванием по правой границе, значение выводится с 2 знаками после десятичного разделителя.

28. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = exp(-x_n/m)sin(mx_n)cos(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 10 символов с выравниванием по правой границе, значение функции выводится с 4 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл сначала переставленные в обратном порядке отрицательные, затем переставленные в обратном порядке положительные числа каждой строки, при этом колонки должны иметь ширину 8 символов с выравниванием по правой границе, значение выводится с 3 знаками после десятичного разделителя.

29. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = exp(-x_n/m)sin(mx_n^2)cos(mx_n), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 12 символов с выравниванием по правой границе, значение функции выводится с 6 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл сначала переставленные в обратном порядке удвоенные отрицательные, затем переставленные в обратном порядке положительные числа каждой строки, при этом колонки должны иметь ширину 9 символов с выравниванием по правой границе, значение выводится с 4 знаками после десятичного разделителя.

30. Вариант Напишите программу, которая запрашивает у пользователя два целых числа N и M и два вещественных числа x_0 и x_N и записывает в текстовый файл таблицу значений функции $y_m(x_n) = exp(-x_n/m)sin(mx_n)cos(mx_n^2), n = \overline{1,N}, m = \overline{1,M}, x \in [x_0, x_N]$ (N строк, M колонок, колонки имеют ширину 10 символов с выравниванием по правой границе, значение функции выводится с 5 знаками после десятичного разделителя), закрывает файл. Затем открывает этот же файл на чтение и переписывает в другой файл сначала переставленные в обратном порядке отрицательные, затем переставленные в обратном порядке утроенные положительные числа каждой строки, при этом колонки должны иметь ширину 9 символов с выравниванием по правой границе, значение выводится с 4 знаками после десятичного разделителя.

Глава 8. Использование символов и строк.

В языке программирования С функции для работы со строками объявляются в заголовочном файле *string.h*, а функции работы с символами в файле *ctype.h*. Нужно не забывать подключать их.

Следует обратить внимание на то, что строкой в С считается любая последовательность символов, оканчивающаяся нулевым кодом. Поэтому функциям можно передавать адрес любого символа строки, и они будут считать строкой оставшуюся часть. Это свойство можно эффективно использовать. Для примера вставим одну строку в середину другой:

```
char s1[20] = "one three", s2[20] = "two";
strcpy(s2+3, s1+3);
strcpy(s1+4, s2);
puts(s1);
```

Здесь сначала во вторую строку копируется конец первой, получается "two three". Затем в первую строку, минуя ее начало, копируется вторая.

Еще одной интересной функцией является *strtok*(). С ее помощью можно разбить строку на отдельные части (лексемы). Однако, эта функция портит исходную строку, вставляя в неё терминальные нули. Поэтому, если исходная строка нужна в последующем коде, то необходимо сделать её копию.

Рассмотрим пример:

```
#include <stdio.h>
#include <string.h>
int main()
{
	char str[200],str2[200];
	fgets(str2,200,stdin); // ввод строки из консоли
	strcpy(str,str2); // делаем копию для работы с strtok
	// Объявляем с запасом массив из 30 слов длиной не более 19 символов
	char words[30][20];
	char * pch = strtok (str," ,.-"); // получаем первое слово
	// во втором параметре указаны разделители слов (пробел, запятая, точка, тире)
	int n=0;
	strcpy(words[n++],pch); // копируем первое слово

	// пока есть очередное слово и место в массиве words:
	while( (pch = strtok (NULL, " ,.-")) != NULL && n<30)
		strcpy(words[n++],pch);// копируем очередное слово
	for(int i=0;i<n;i++)
	    printf("%s %d\n",words[i],strlen(words[i])); // выводим слова и их длины
	return 0;
}
```

При первом вызове *strtok*() в функцию передается указатель на первый символ массива и строка-разделитель. После этого вызова массив *str* изменяется, в нем остается только первое слово, также функция возвращает указатель на это слово, который присваивается *pch*. Остаток массива потерялся, но внутри *strtok*() сохраняется указатель на этот остаток. Когда в качестве адреса передается NULL, то функция работает с этим остатком.

Для подсчета количества одинаковых символов строки можно использовать их коды для индексирования элементов массива, в которых будет храниться количество соответствующих символов. Обратите внимание, что тип char – это знаковый целочисленный тип в диапазоне от -128 до +127. Поэтому, для индексации массива надо использовать unsigned char (диапазон от 0 до 255).

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void CountChars(unsigned char* str,int* freq)
{
	int i = 0;
	for(i=0;i<256;i++)
		freq[i]=0; // обнуляем массив
// подсчет количества символов равных str[i]:
	for(i=0;i<str[i]!=0;i++)
		freq[str[i]]++;
}
void Print(int* freq)
{
	int i;
	for(i=0;i<256;i++)
```

```
            if(freq[i]!=0) // если символ встречался в строке
// вывод кода символа и сколько раз встретился:
                  printf("%c %d\n",i,freq[i]);

}
int main()
{
      unsigned char myString[100];
      int f[256];

      system("chcp 1251"); // установка русской кодировки в консоли

      printf("Введите строку:\n");
      fgets((char*)myString,100,stdin);
      CountChars(myString,f);
      Print(f);
      getchar();
      return 0;
}
```

Здесь функция `CountChars()` выполняет подсчет встречаемости символов, а функция `Print()` выводит результат. Обратите внимание на выражение *freq*[*str*[*i*]]++, в котором код символа *str*[*i*] является индексом для массива *freq*. Это позволяет подсчитать количество символов, имеющих данный код.

Варианты задач для решения

В задании необходимо написать функцию соответствующего варианта и в функции *main()* протестировать ее, вводя необходимые строки из файла, а дополнительные данные из консоли, выводя результат на экран.

1. Вариант Написать функцию, которая удаляет из строки все знаки пунктуации. Для проверки знаков использовать библиотечную функцию *ispunct*().
2. Вариант Написать функцию, которая проверяет, является ли строка целым десятичным числом с опциональным знаком. Для проверки цифр использовать библиотечную функцию *isdigit*().
3. Вариант Написать функцию, которая подсчитывает количество гласных букв в строке используя массив гласных букв и функцию *strcspn*().
4. Вариант Написать функцию, которая подсчитывает, сколько раз в строке встречается заданная буква используя функцию *strchr*() и после этого удалить ее из строки.
5. Вариант Написать функцию, которая заменяет в строке s все вхождения подстроки s1 на подстроку s2 используя функцию *strstr*().
6. Вариант Написать функцию, которая находит все слова строки, отличные от последнего, используя функции *strtok*() и *strcmp*().
7. Вариант Написать функцию, которая удаляет из строки слова начинающиеся на указанную букву используя функцию *strtok*().
8. Вариант Написать функцию, которая определяет количество различных слов строки используя функции *strtok*() и *strcmp*().
9. Вариант Написать функцию, которая вычисляет количество вхождений переданного символа в строке используя функцию *strchr*().
10. Вариант Написать функцию, проверяющую, что строка является палиндромом используя функции *strrev*() и *strcmp*().

11. Вариант
Написать функцию, вычисляющую количество цифр в строке используя функцию *isdigit*().

12. Вариант
Написать функцию, вычисляющую длину слов в строке используя функцию *strtok*().

13. Вариант
Написать функцию, находящую позицию самого последнего вхождения указанного символа в строке используя функцию *strchr*() и сравнить с результатом функции *strrchr*().

14. Вариант
Написать функцию сравнения двух строк без учета регистра используя функцию *_stricmp*(). Функция возвращает 0, если строки одинаковые и 1, если разные.
Не забудьте задать русскую локаль вызовом функции *setlocale*(), если строки содержат кириллицу.

15. Вариант
Написать функцию, инвертирующую все слова переданной строки.

16. Вариант
Написать функцию, которая принимает на вход строку и два символа. Функция заменяет все вхождения первого символа в строке на второй символ используя функцию *strchr*().

17. Вариант
Написать функцию, проверяющую, что переданную строку можно разбить на две одинаковые подстроки используя функцию *strncmp*().

18. Вариант
Написать функцию, которая записывает строковое представление числа в массив. Число и массив передаются ей в качестве параметров. Сравнить ее с функцией *itoa*().

19. Вариант
Написать функцию, которая находит самое короткое слово в строке используя функцию *strtok*(). Слова отделяются пробелами.

20. Вариант
Написать функцию, которая выполняет выравнивание строки, введенной пользователем, до заданной ширины используя функцию *strtok*() и добавляя необходимое количество пробелов.

21. Вариант
Написать функцию, выводящую на экран слова, которые одновременно содержатся в каждой из двух заданных строк используя функции *strtok*() и *strcmp*().

22. Вариант
Написать функцию, которая используя функции *strtok*() и *strcmp*() выводит на экран слова, которые являются общими для трех строк, а также для каждой пары строк.

23. Вариант
Написать функцию, которая используя функции *strtok*() и *strcmp*() находит слова, которые повторяются два или более раз и выводит их с указанием числа повторений. В случае отсутствия таких слов выводится соответствующее сообщение.

24. Вариант
Написать функцию, которая используя функции *strtok*() и *strchr*() определяет количество слов в заданной строке, содержащих заданную букву. Вывести число слов в строке, содержащих русские буквы ‘*a’* и ‘*p’*. В случае отсутствия таких слов вывести соответствующее сообщение.

25. Вариант
Написать функцию, которая используя функции *strtok*() и *strchr*() выводит все слова заданной строки, в которые заданная буква входит не менее K раз. В случае отсутствия таких слов выводится соответствующее сообщение.

26. Вариант
Написать функцию, которая используя функции *strtok*() и *strcmp*() вычисляет количество разных слов в заданной строке.

27. Вариант
Написать функцию, которая преобразует целое число(int) в строку, представляющую число в указанной системе счисления (от 2 до 10).

28. Вариант

Написать функцию возвращающую максимальное количество подряд идущих произвольных символов в строке. Например, для строки "asdaajhys as asaaaafg r" нужно возвратить 4.

29. Вариант

Написать функцию, которая возвращает символ, встречающийся в строке чаще всего.

30. Вариант

Написать программу, которая выводит на экран слово "yes", если из букв введенной строки X можно составить введенную строку Y, при условии, что каждую букву строки X можно использовать один раз. Если это невозможно сделать, вывести "no". Подсказка: можно сравнить частотные таблицы первой и второй строки.

Глава 9. Использование структур.

Если в программе используются разнотипные данные, имеющие некоторое общее назначение, то имеет смысл объединять их в структуры.

Рассмотрим для примера объект «файл». У него имеется имя, тип (расширение), размер. Кроме того, чтобы найти его на диске необходимо знать имя диска, и папки, в которой он находится. Эту информацию можно объединить в структуру следующим образом:

```
struct File
{
	char name[80]; // Имя
	char type[16]; // Тип
	char disk; // буква диска
	char dir[100]; // директория
	long size;   // размер файла
};
```

Рассмотрим программу, выполняющую основные операции с данными структурного типа. Структуры в файл целесообразно писать в бинарном режиме, т.к. в этом случае не требуется подробное перечисление элементов структуры. Однако, следует учесть, что такой файл нельзя будет прочитать текстовым редактором. Кроме того, в составе структур не должно быть элементов с динамическим выделением памяти (иначе в файл будут записываться указатели на них, а не сами данные). Такая запись слегка избыточна, т.к. в файл будут записываться структуры целиком, включая неиспользованные участки текстовых массивов. Но с ней можно смирится, если записей не слишком много.

Рассмотрим пример программы, обрабатывающей данные о файлах.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct File
{
	char name[80]; // Имя файла
	char type[16]; // Тип файла
	char disk; // буква диска
	char dir[100]; // директория
	long size;   // размер файла
};
const int MAX_SIZE=100; // максимальное к-во обрабатываемых структур

void Input(File* pf)// функция ввода структуры, pf - адрес куда вводить
{
	printf("Имя: ");
	fgets(pf->name, 80, stdin); // ввод строки с именем файла
	pf->name[strlen(pf->name)-1]='\0';// удаление из строки символа '\n'
	printf("Тип: ");
	fgets(pf->type, 16, stdin); // ввод строки с названием типа файла
	pf->type[strlen(pf->type)-1]='\0';// удаление из строки символа '\n'
	printf("Диск: ");
	pf->disk=getchar(); // ввод символа - имени диска
	getchar(); // удаление из входного потока символа '\n'
	printf("Директория: ");
	fgets(pf->dir, 100, stdin); // ввод строки с директорией
	pf->dir[strlen(pf->dir)-1]='\0';// удаление из строки символа '\n'
	printf("Размер: ");
	scanf("%ld",&pf->size); // ввод размера файла
	getchar(); // удаление из входного потока символа '\n'
}
void Print(File f) // функция вывода структуры, f - выводимая структура
{
	printf("\nИмя: %s\n", f.name);
	printf("Тип: %s\n",f.type);
	printf("Диск: %c\n",f.disk);
	printf("Директория: %s\n",f.dir);
	printf("Размер: %d байт\n",f.size);
}
void Write(File f[],int n) // функция записи в файл всех структур (n шт.)
{
```

```
	FILE* store;
	store=fopen("files.dat","wb"); // открытие на запись в бинарном режиме
	if(store!=NULL)
	{
		fwrite((void*)f,sizeof(File),n,store);
		fclose(store);
		printf("Записано %d структур\n",n);
	}
	else
		puts("Файл для записи не открылся.");

}
int Read(File* f) // функция чтения из файла всех структур
{
	FILE* store;
	store=fopen("files.dat","rb"); // открытие на чтение в бинарном режиме
	if(store!=NULL)
	{
		int n=0;
		while( n < MAX_SIZE &&
			fread((void*)(f+n),sizeof(File),1,store)==1 )
			// читаем структуры по одной, пока не достигнем конца файла
			n++;
		printf("Прочитано %d структур\n",n);
		return n;
	}
	else
	{
		puts("Файл для чтения не открылся.");
		return 0;
	}
}
void FullName(char* fname,File f) // функция формирующая полное имя файла
{
	sprintf(fname,"%c:\\%s\\%s.%s",f.disk,f.dir,f.name,f.type);
}
int main()
{
	system("chcp 1251");

	File f1[MAX_SIZE]; // оъявление массива структур
	int n=0; // счетчик количества заполненных структур
	int action; // переменная выбора пункта меню
	do
	{
		puts("\nВыберите действие\n"
			"1 - Ввод с клавиатуры\n"
			"2 - Вывод на экран\n"
			"3 - Запись\n"
			"4 - Чтение\n"
			"5 - Полные имена\n"
			"0 - Выход из программы\n"
			);
		scanf("%d",&action);
		getchar(); // удаление из потока символа "Enter"
		switch(action)
		{
		case 1: //  Ввод с клавиатуры одной структуры
			Input(&f1[n]); // указываем адрес, куда вводить
			n++;
			break;
		case 2: // Вывод на экран всех структур
			for(int i=0;i<n;i++)
				Print(f1[i]); // указываем, что выводить
			break;
```

```
            case 3: // запись всех структур на диск
                  Write(f1,n);
                  break;
            case 4: // чтение всех структур с диска
                  n=Read(f1); // n - прочитанное к-во
                  break;
            case 5: // обработка
                  for(int i=0;i<n;i++)
                  {
                        char FileFullName[100];
                        FullName(FileFullName,f1[i]);
                        // вывод полного имени и размера
                        printf("%s %ld\n",FileFullName,f1[i].size);
                  }
                  break;
            }
      }while(action!=0); // завершаем цикл при вводе нуля
      return 0;
}
```

Функция *Input*() вводит данные очередной структуры с клавиатуры, функция *Print*() выводит содержимое структуры на экран, функция *write*() выводит массив структур в бинарный файл, функция *Read*() читает массив структур из бинарного файла и возвращает количество прочитанных структур, функция обработки *FullName*() из полей структуры формирует полное имя, включающее букву диска, имя папки и имя файла.

В основной программе имеется цикл, позволяющий выбрать необходимое действие с массивом структур.

Варианты задач для решения

В задании необходимо написать программу, которая содержит функции ввода и вывода информации структурного типа с клавиатуры и из файла в бинарном режиме (аналогично приведённому образцу) и функции, выполняющие обработку данных структуры в соответствии с решаемым вариантом.

В функции *main*() необходимо реализовать консольное меню, аналогичное приведённому выше образцу, позволяющее считывать данные из файла в массив структур, записывать массив в файл, добавлять данные с клавиатуры, выводить содержимое всего массива, выполнять указанные действия с этим массива.

1. Вариант

Задан список структур Books: имя и фамилия писателя, название произведения, год издания.
Написать программу, которая
-по заданному автору выводит все его произведения
-выводит список всех авторов, произведения которых начинаются с заданной буквы.

2. Вариант

Задан список пользователей User: фамилия, имя, код пользователя, дата регистрации.
Написать программу, которая выводит
-всех пользователей фамилии которых начинаются с заданной буквы
-всех тезок (если они есть).

3. Вариант

Автомагазин продает автомобили. О каждой машине хранится информация о марке автомобиля, фирме изготовителе, годе выпуска, объеме двигателя, типе кузова, стоимости.
Написать программу, которая выводит список
- всех автомобилей указанной фирмы и имеющих заданный тип кузова
- автомобилей стоимостью ниже заданной выпущенных после указанного года.

4. Вариант

Задан список структур Mountains для представления информации по горным вершинам: название территории, название вершины и ее высота.
Написать программу, которая выводит
-список всех вершин расположенных на заданной территории
-названия всех вершин имеющих высоту выше среднего значения.

5. Вариант

Задан список структур Hotels описывающий гостиничный номер:
название гостиницы, номер, комфортность (люкс, полулюкс, стандарт,
эконом), количество человек, стоимость.
Написать программу, которая выводит информацию
- о номерах, в названии гостиницы которых есть по 2 буквы 'a'
- о номерах, в которых стоимость люкса не более заданной.

6. Вариант

Задан список структур Student, содержащую фамилию и инициалы, номер группы и успеваемость (массив из пяти элементов).
Написать программу, которая выводит
-фамилии и номера групп студентов, имеющих хотя бы одну неудовлетворительную оценку;
- фамилии и успеваемость студентов из заданной группы.

7. Вариант

Задан список структур Sklad (вид товара; наименование товара; количество; стоимость; процент торговой надбавки).
Написать программу, которая выводит
- суммарное количество товаров каждого вида
- список товаров заданного вида стоимостью не выше заданной.

8. Вариант

Задан список структур Saity: URL, E-mail, ответственный, дата внесения, количество обращений.
Написать программу, которая выводит
- список сайтов с датой внесения не позже указанной и количеством обращений меньших заданного
- количество сайтов зарегистрированных за каждым из ответственных.

9. Вариант

Задан список структур Journals: код, название, периодичность, цена, дата начала издания. Написать программу, которая выводит
- суммарное количество журналов с периодичностью 1 месяц
- список журналов стоимостью не выше заданной и датой начала издания не ранее указанной.

10. Вариант

Задан список структур Books: ISSN, название, автор, к-во страниц, год издания, издательство. Написать программу, которая выводит
- количество книг у каждого из авторов.
- список книг с количеством страниц не менее заданного и годом издания не ранее указанного.

11. Вариант

Задан список структур Films: код, название, режиссер, дата выхода, кассовые сборы. Вывести
- суммарное количество фильмов каждого из режиссеров
- список фильмов с кассовыми сборами не менее заданного вышедших за указанный год.

12. Вариант

Задан список структур Songs: группа, солист, название, дата, рейтинг.
Написать программу, которая выводит
- количество песен у каждого из солистов
- список песен за указанный год с рейтингом не ниже заданного.

13. Вариант

Задан список структур Oborudovanie: Инв. номер, наименование, дата приобретения, стоимость, аудитория.
Написать программу, которая выводит
- суммарное количество оборудования по каждому из наименований
- список оборудования стоимостью не выше заданной в указанной аудитории.

14. Вариант

Задан список структур Materialy: артикул, наименование, дата приобретения, дата списания, стоимость.
Написать программу, которая выводит
- суммарное количество материалов каждого наименования
- список материалов со стоимостью не выше заданной и списанных за указанный год.

15. Вариант

Задан список структур Tovar: наименование, артикул, штрих код, производитель, единица измерения, цена.

Написать программу, которая выводит

- суммарное количество товаров каждого производителя
- список товаров данного производителя стоимостью не выше заданной.

16. Вариант

Задан список структур Pokupateli: код покупателя, фамилия, адрес, телефон, сумма покупки. Написать программу, которая выводит

- общую сумму покупок по каждому покупателю
- список покупателей делающих покупки стоимостью не ниже заданной.

17. Вариант

Задан список структур Postavshiki: код поставщика, наименование, адрес, телефон, вид товара. Написать программу, которая выводит

- суммарное количество товаров каждого поставщика
- список поставщиков заданного вида товара.

18. Вариант

Задан список структур Zakazy: код заказа, дата заказа, код заказчика, артикул товара, количество товара.

Написать программу, которая выводит

- суммарное количество заказанных предметов по каждому из артикулов
- список заказов за указанный месяц по указанному артикулу.

19. Вариант

Задан список структур Prodazhi: дата покупки, код-заказа, наименование товара, количество, вид оплаты(нал/карта).

Написать программу, которая выводит

- суммарное количество покупок каждого из товаров за указанный год
- список покупок с количеством не ниже заданного оплаченных наличными.

20. Вариант

Задан список структур Postavki: дата поставки, артикул, количество, цена-поставки.

Написать программу, которая выводит

- суммарное количество поставок по каждому из артикулов
- список поставок с ценой не ниже заданной за указанный год.

21. Вариант

Задан список структур Fishing: вид рыбы, дата ловли, вес, длина, занятое место.

Написать программу, которая выводит

- суммарное количество выловленной рыбы по каждому из месяцев
- список выловленной рыбы весом не ниже заданного для заданного вида рыбы.

22. Вариант

Задан список структур Turnir: дата, код участника, фамилия, к-во выигрышей, проигрышей, ничьих, место.

Написать программу, которая выводит

- суммарное количество выигрышей
- список участников занимавших первые места в заданном году.

23. Вариант

Задан список структур Futbol: код команды, название, к-во забитых мячей, пропущенных мячей, место.

Написать программу, которая выводит

- суммарное количество забитых мячей
- список команд занимавших призовые места.

24. Вариант

Задан список структур Electrichestvo: код абонента, адрес, дата, показание пред, показание тек, уплачено.

Написать программу, которая выводит

- суммарное количество потраченной энергии по каждому из абонентов
- список должников на указанную дату.

25. Вариант

Задан список структур Kvartplata: лицевой счет, фамилия, адрес, телефон, дата оплаты, сумма. Написать программу, которая выводит

- суммарную суммы оплаты по каждому абоненту
- список абонентов заплативших в указанном году сумму не менее заданной.

26. Вариант

Задан список структур Account: номер счета, пароль, фамилия, сумма, процент, срок хранения. Написать программу, которая выводит

- общую сумму на счетах у каждого из клиентов
- список счетов имеющих срок хранения не менее заданного под указанный процент.

27. Вариант

Задан список структур User: Фамилия, логин, пароль, дата, суммарный трафик. Написать программу, которая выводит

- общий трафик каждого пользователя за указанный месяц.
- список пользователей, у которых трафик за указанную дату был не менее заданного.

28. Вариант

Задан список структур Kafedra: код, название, зав-кафедрой, аудитория, телефон.
Написать программу, которая выводит

- общее количество аудиторий закрепленных за каждой кафедрой
- список кафедр пользующихся указанным телефоном.

29. Вариант

Задан список структур Univer: факультет, декан, телефон, количество кафедр, количество учебных групп.
Написать программу, которая выводит

- суммарное количество учебных групп
- список факультетов имеющих количество кафедр не более заданного и количество групп не менее заданного.

30. Вариант

Задан список структур Person: фамилия, имя, отчество, дата рождения, семейное положение, адрес проживания.
Написать программу, которая выводит

- суммарное количество записей по каждому из семейных положений.
- список тех, кто родился не ранее указанной даты и имеет указанное имя.

Глава 10. Зачет: методика его проведения, теоретический минимум и типовые задания.

Зачет проводится в практикуме и состоит из двух частей: теоретической и практической.

Теоретический минимум

Проверка теоретических знаний производится в виде опроса по любым вопросам, входящим в программу первого семестра и теоретический минимум. Вопрос, как правило, формулируется в виде просьбы написать на листе бумаги короткий (2-5 строк) код, иллюстрирующий то или иное понятие языка программирования. Обычно задается от двух до пяти вопросов. Студенты, не сдавшие теоретическую часть, автоматически не допускаются к выполнению практической части зачета.

Ниже приводится перечень сведений о языке программирования из теоретического минимума знаний, которые студент должен иметь для получения зачета.

1. Основы синтаксиса языка Си, структура консольного приложения.
2. Фундаментальные типы данных (**`bool, char, int, double`**).
3. Определение переменных и констант.
4. Оператор **`sizeof()`**.
5. Выражения, операции, комментарии.
6. Оператор приведения типа.
7. Операторы инкремента и декремента.
8. Приоритет операторов в выражениях.
9. Блоки и правила видимости переменных.
10. Условный оператор и оператор перехода (**`if, goto`**).
11. Оператор множественной альтернативы (**`switch`**).
12. Цикл **`while`**. Прерывание цикла. Переход к следующей итерации.
13. Цикл **`do ... while`**. Прерывание цикла. Переход к следующей итерации.
14. Цикл **`for`**. Прерывание цикла. Переход к следующей итерации.
15. Математические функции стандартной библиотеки Си (**`<math.h>`**).
16. Форматированный консольный ввод (**`<stdio.h>`**): параметры функции **`scanf()`**.
17. Форматированный консольный вывод (**`<stdio.h>`**): параметры функции **`printf()`**.
18. Форматированный файловый ввод-вывод (**`<stdio.h>`**).
19. Бесформатный файловый ввод-вывод (**`<stdio.h>`**).
20. Массивы. Передача массивов в параметрах функции.
21. Определение функции. Прототип функции. Рекурсия.
22. Параметры функции **`main()`**.
23. Раздельная компиляция программных модулей. Использование **`*.h`** файлов.
24. Внешние (**`extern`**) и глобальные переменные.
25. Статические (**`static`**) переменные.
26. Статические (**`static`**) функции.
27. Указатели и операторы, с ними связанные.
28. Указатель на функцию.
29. Функции для работы с динамической памятью **`malloc()/realloc()/free()`**.
30. Строки Си. Функции для работы со строками (**`<string.h>`**).
31. Функции для работы с символами (**`<ctype.h>`**).
32. Пользовательский тип данных **`enum`**.
33. Пользовательский тип данных **`struct`**.
34. Пользовательский тип данных **`union`**.
35. Определение синонимов типов (**`typedef`**).
36. Директивы препроцессора для условной компиляции и их использование.
37. Директивы препроцессора для включения файлов и их использование.
38. Макроопределения препроцессора (с параметрами и без).

Вопросы по теоретической части

Здесь приводятся примерные варианты билетов с вопросами к теоретическому минимуму, на которые студент должен уметь отвечать на зачете.

1. Вариант

1. Какие базовые типы данных Вы знаете? Сколько места они занимают в памяти компьютера?
2. Что означает команда «continue» и где она применяется?
3. Выделите память под вещественный массив чисел двойной точности из 123 элементов.

2. Вариант

1. Какие операторы языка Си Вы знаете? Какой у них приоритет?
2. Какие виды циклов Вы знаете? Чем они отличаются друг от друга?
3. Что такое прототип функции и где он используется?

3. Вариант

1. Что такое операции постфиксного инкремента и префиксного декремента?
2. Что означает команда «break» и где она применяется?
3. Что такое рекурсивная функция? Приведите пример.

4. Вариант

1. Что означает строка:
   void func(int,int*);
2. Что такое метка и где она используется?
3. Что такое цикл «с предусловием» и цикл «с постусловием»?

5. Вариант

1. Сколько места в памяти компьютера занимают вещественные переменные и переменные с двойной точностью? Что такое точность представления вещественного числа?
2. Что означает команда «goto» и где она применяется?
3. Как найти заданную подстроку в другой строке?

6. Вариант

1. Что такое массив и что такое указатель? Приведите примеры.
2. Верно ли синтаксически написана программа:
   void main( ) { }
3. Какие функции из библиотеки <string.h> Вы знаете?

7. Вариант

1. Что такое операция получения адреса? Где она используется?
2. Какие функции ввода-вывода Вы знаете?
3. Что такое перечислимый тип? Как он определяется?

8. Вариант

1. Как получить значение по указателю? Приведите примеры.
2. Какие форматирующие последовательность функции printf() Вы знаете?
3. Опишите структуру, состоящую из символьного массива из 12 членов, целого и вещественного числа.

9. Вариант

1. В каком случае можно изменить размер ранее выделенного массива? Как это сделать?
2. Напишите цикл do ... while(), меняющий порядок элементов массива на противоположный.
3. Для чего нужен оператор typedef?

10. Вариант

1. Напишите цикл for, меняющий порядок элементов массива на противоположный.
2. Как получить указатель на переменную?
3. Какие функции для работы с отдельными символами Вы знаете?

11. Вариант

1. Как записать массив в бинарный (неформатированный) файл?
2. Какие значения будут иметь все переменные в результате выполнения программы:
   int a = 1, b = 2, c = 3;
   c -= a += b = c;
3. Как вернуть из функции массив?

12. Вариант

1. Откройте файл "myfile.txt" на чтение.
2. Какие значения будут иметь все переменные в результате выполнения программы:
int a = 1, b = 2;
a += b++;
3. Что такое union?

13. Вариант

1. Как передать функцию в другую функцию? Как создать массив функций?
2. Как с помощью функции языка Си прочитать в три переменные три вещественных числа, разделенных пробелами?
3. Как обратиться к функции, определенной в другом файле с исходным текстом?

14. Вариант

1. Что такое заголовочный файл?
2. Как определить размер вещественного массива в байтах?
3. Напишите функцию, которая складывает два вещественных числа и возвращает результат.

15. Вариант

1. Для чего нужны статические переменные? Чем они отличаются от глобальных?
2. Что такое макроопределение с параметром? Приведите пример.
3. Как распечатать на экран строки из текстового файла, открытого на чтение?

16. Вариант

1. Как найти длину текстовой строки?
2. Какие значения будут иметь все переменные в результате выполнения программы:
int a = 1, b = 4, c = 3;
a = --b > c;
3. Для чего применяется оператор default?

17. Вариант

1. Как реализовать компиляцию некоторой функции в зависимости от условия?
2. Напишите циклы while, транспонирующие матрицу.
3. Опишите структуру, состоящую из имени, фамилии и года рождения студента.

18. Вариант

1. Определите двумерный вещественный массив.
2. Какие значения будут иметь все переменные в результате выполнения программы:
int a = 1; int* b = &a;
(*b)++;
3. Определите массив из трех структур, которые представляют разные точки на плоскости.

19. Вариант

1. Как изменить значение глобальной переменной, заданной в другом файле исходного текста проекта?
2. Какие форматирующие последовательность функции printf()Вы знаете?
3. Как задать перечислимый тип из трех убывающих элементов?

20. Вариант

1. Выделить динамическую память под строку из 20 символов.
2. Как передать аргументы с командной строки в функцию main()?
3. Распечатайте число π с тремя знаками после десятичной точки.

21. Вариант

1. Как закрыть функцию от использования в других исходных текстах проекта?
2. Какое значение будет иметь переменная x:
double x = 4;
x *= 1/2;
3. Определите функцию, которая возвращает сумму элементов переданного массива.

22. Вариант

1. Как найти в строке любой из символов, входящих в строку "abc"?
2. Как напечатать целое число с обязательным выводом знака «плюс» для положительных чисел? Как сделать выравнивание по левой границе поля для этого числа?
3. Как принудительно преобразовать тип целого числа к вещественному типу?

23. Вариант

1. Как получить адрес третьего элемента вещественного массива? Если этот адрес уменьшить на единицу – куда он будет указывать?
2. Какое значение получит переменная z в результате выполнения программы:
int a = 1, b = 2, c = 3, d = 4; bool z;
z = a < b || b < c && d < c;
3. Как определить, является ли заданный символ пробельным?

24. Вариант

1. Создайте макроопределение, которое вычисляет сумму двух своих аргументов.
2. Что делает функция scanf()?
3. Откройте файл "myfile.txt" на запись в режиме «запись в конец файла».

25. Вариант

1. Как проверить, открылся ли файл?
2. Какие значения будут иметь все переменные в результате выполнения программы:
int a = 0, b = 1, c = 1;
a = b << c++;
3. Как обратиться к переменной, определенной в ином исходном тексте проекта?

26. Вариант

1. Как объявить и инициализировать указатель на переменную?
2. Какие форматирующие последовательность функции printf() Вы знаете?
3. Объявите структуру, состоящую из даты (3 числа) и названия месяца (массив символов).

27. Вариант

1Как выделить память под двумерный динамический массив?
2. Напишите цикл while(), меняющий порядок элементов массива на противоположный.
3. Приведите пример использования перечисления enum.

28. Вариант

1. Напишите цикл for, находящий последний минимальный элемент массива.
2. Как передать в функцию переменную по указателю?
3. Как объединение union располагает элементы в памяти?

29. Вариант

1. Как прочитать содержимое бинарного (неформатированного) файла с целыми числами?
2. Какие значения будут иметь все переменные в результате выполнения программы:
int a = 1, b = 2, c = 4;
c -= a += b++;
3. Как передать из функции созданный в ней динамический массив?

30. Вариант

1.Как добавить данные в уже имеющийся файл?
2. Какие значения будут иметь все переменные в результате выполнения программы:
int a = 1, b = 2;
a += ++ b;
3. Приведите пример использования объединения union.

Практические задания

Практические навыки программирования проверяются на одной типовой задаче, подобной тем, что выполнялись в течение семестра. Студент должен ее выполнить в течение одной пары (два академических часа) от начала до конца.

Рекомендуется основную массу заданий выдавать по образцу, приведённому в девятой главе: написать программу, которая читает данные (например, матрицу чисел произвольной размерности) из одного файла, как-то преобразует прочитанные данные (например, транспонирует прочитанную матрицу) и записывает получившийся результат в другой файл.

В приводимых ниже типовых заданиях для зачета необходимо написать законченную программу, которая тестирует заданную функцию. Это примеры зачетных заданий, но на реальном зачете могут быть даны другие, примерно такой же сложности.

1. Вариант Функция записывает в файл все четырехзначные натуральные числа из диапазона (2000 – 3000), в записи которых нет двух одинаковых цифр, подсчитывает количество таких чисел, возвращает в **`main()`**.
2. Вариант Функция считывает из файла вещественный массив неизвестной длины (до конца файла). Надо найти в массиве два элемента, модуль разности которых имеет наименьшее значение. Напечатать эти элементы и их индексы.
3. Вариант Функция, работающая как простейший калькулятор, выполняющий действия «**+**», «**–**», «*****», «**/**» над двумя целыми массивами одного размера, считанными из файла. Результат записать в файл.
4. Вариант Функция считывает из файла целый массив неизвестной длины (до конца файла), находит в массиве минимальный по модулю элемент и заменяет им все элементы массива с четными номерами. Измененный массив записывается в файл.
5. Вариант Функция считывает из файла целый массив неизвестной длины (до конца файла). Необходимо найти максимальный по модулю элемент этого массива и заменить им все нулевые элементы массива. Подсчитать и вернуть в **`main()`** число таких элементов. Измененный массив сохранить в другой файл.
6. Вариант Функция считывает из файла целый массив неизвестной длины (до конца файла). Определить и напечатать три наибольших элемента этого массива. Подсчитать их сумму, вернуть в **`main()`**.
7. Вариант Функция считывает из файла вещественный массив неизвестной длины (до конца файла), находит в массиве минимальный по модулю элемент и заменяет им все элементы с нечетными номерами. Измененный массив записывается в файл.
8. Вариант Функция получает два целых числа, m и n, создает массив из простых чисел, расположенных в интервале от m до n, подсчитывает их количество, массив записывает в файл.
9. Вариант Функция считывает из файла целый массив неизвестной длины (до конца файла), запрашивает у пользователя целое число **`k`** и находит элемент массива, значение которого наиболее близко к введенному числу и возвращает его индекс.
10. Вариант Функция, вычисляющая методом трапеций определённый интеграл от непрерывной положительной ***`f(x)`*** на отрезке **`[a, b]`**. Шаг разбиения уменьшать, пока площадь от итерации к итерации изменяется более, чем на **`ε`**. Параметры **`a`**, **`b`** и **`ε`** передаются из ***`main()`***, куда и возвращается результат.

11. Вариант

Функция, вычисляющая методом верхних прямоугольников определённый интеграл от непрерывной положительной ***f(x)*** на отрезке **[a, b]**. Шаг разбиения уменьшать, пока площадь от итерации к итерации изменяется более, чем на **ε**. Параметры **a**, **b** и **ε** передаются из **main()**, куда и возвращается результат.

12. Вариант

Функция, вычисляющая методом деления отрезка пополам с точностью эпсилон корень уравнения **f(x) = 0** на отрезке **[a, b]**. Параметры **a**, **b** и **ε** передаются из ***main()***, куда и возвращается результат. Функция **f(x)** вычисляется в отдельном блоке.

13. Вариант

Функция, которая считывает массив чисел неизвестной размерности из файла, сортирует его и записывает в другой файл. Функция принимает название входного и выходного файла. Необходимо использовать функции выделения динамической памяти.

14. Вариант

Функция, которая считывает из файла численные данные, записанные в 2 колонки, и переписывает их в другой файл в строку через запятую. Необходимо использовать функции выделения динамической памяти.

15. Вариант

Функция, которая находит в текстовом файле заданную пользователем строчку и указывает номер строки файла, в которой строка была найдена.

16. Вариант

Функция, подсчитывающая количество букв в тексте, находящемся в указанном файле. Функция принимает название входного файла и возвращает количество букв (цифры и пунктуацию не учитывать).

17. Вариант

Функция, обрабатывающая текстовый файл. В файле имеется массив x координат, размер которого заранее не известен. Функция считывает эти значения и записывает в выходной файл **x** и ***sin(x/2)*** в 2 колонки. Необходимо использовать функции выделения динамической памяти.

18. Вариант

Функция, обрабатывающая текстовый файл. В файле имеются две колонки вещественных чисел (пары координат **x**, **y**). Надо отсортировать пары чисел по координате **x** и записать отсортированные таким образом две колонки в другой файл.

19. Вариант

Имеется 2 текстовых файла, в которых записаны два n-мерных вектора. Значение **n** заранее неизвестно. Создать функцию, которая считывает вектора из файлов, производит сложение или поэлементное умножение векторов и записывает результат в 3-й файл. Функция принимает название входного и выходного файлов и знак операции («**+**» или «*****»). Необходимо использовать функции выделения динамической памяти.

20. Вариант

Функция, перемножающая две матрицы вещественных чисел произвольного размера. Каждая матрица записана в своем текстовом файле. Функция должна считывать эти матрицы, перемножать их и записывать результат в третий файл. Если размерности матрицы не позволяют выполнить умножение (количество столбцов первой матрицы не равно количеству строк второй матрицы), то вывести соответствующее сообщение. Функция принимает название входных и выходного файлов.

21. Вариант

Создать структуру, представляющую собой рациональное число (пара целых чисел: числитель и знаменатель). Написать функции для выполнения четырех арифметических действий с такими структурами. Написать тестовую программу, в которой нужное арифметическое выражение вводит пользователь.

22. Вариант

Написать функцию, которая считывает из указанного ей файла матрицу произвольной размерности, считает среднее арифметическое и стандартное отклонение для каждой колонки этой матрицы и результат распечатывает на экране.

23. Вариант

Написать функцию, которая сортирует в алфавитном порядке переданный ей массив англоязычных текстовых строк. Строки считываются из указанного файла, а результат выводится на экран.

24. Вариант

Создать структуру, которая будет содержать вещественный динамический вектор произвольной размерности. Написать функцию вставки нового элемента в любое место этого вектора.

25. Вариант

Написать функцию определяющую количество различных чисел матрицы.

26. Вариант

Дан одномерный массив. Сформировать новый массив из элементов десятичное представление которых содержит только чётные цифры (0,2,4,6,8).

27. Вариант

Написать функцию сжимающую массив, удалив из него все элементы, которые находятся между первым и вторым нулевыми элементами.

28. Вариант

Написать функцию, которая в каждой строке матрицы находит наибольший элемент и меняет его местами с элементом главной диагонали.

29. Вариант

Написать функцию вставляющую нулевую строку перед строкой, где находится первый минимальный элемент матрицы.

30. Вариант

Написать функцию находящую произведение тех элементов квадратной матрицы, которые расположены над главной диагональю и сумму элементов, расположенных под побочной диагональю.

31. Вариант

Написать функцию удаляющую из массива все элементы, значения которых больше трех среднеквадратичных отклонений (больше трех сигма).

32. Вариант

Написать функцию определяющую количество «особых» элементов матрицы, считая элемент «особым», если он больше суммы остальных элементов своего столбца.

33. Вариант

Написать функцию создающую массив из порядковых номеров максимальных по модулю элементов в столбцах матрицы.

34. Вариант

Написать функцию оставляющую в массиве только те элементы, которые удовлетворяют двойному неравенству $A_0 < A_k < A_n$. k=1,2,…,(n-1)

35. Вариант

Написать функцию определяющую сумму элементов в тех столбцах матрицы, которые не содержат отрицательных элементов.

36. Вариант

Написать функцию упорядочивающую массив целых положительных чисел по возрастанию методом выбора и возвращающую номер минимального простого числа в полученном массиве.

37. Вариант

Написать функцию, которая добавляет минимальный элемент матрицы к элементам четных столбцов, значения которых находятся в заданном интервале [a,b].

38. Вариант

Написать функцию находящую номер первой строки матрицы, не содержащей ни одного четного элемента, и циклически сдвинуть в ней элементы так, чтобы элемент главной диагонали стал первым элементом этой строки.

39. Вариант

Написать функцию удаляющую из матрицы строку, в которой больше всего отрицательных элементов.

40. Вариант
Написать функцию сжимающую динамический массив с большим количеством нулей. Необходимо в каждой группе нулей первый нуль оставить, второй заменить на кол-во нулей в группе, остальные удалить.

41. Вариант
Написать функцию удаляющую элементы динамического массива, расположенные между его минимальным и максимальным элементами.

42. Вариант
Написать функцию, которая в квадратной матрице находит строку с наибольшей суммой элементов и поменяет её с первым столбцом.

43. Вариант
Написать функцию меняющую местами левую верхнюю и правую нижнюю четверти матрицы.

44. Вариант
Написать функцию удаляющую последний столбец матрицы, содержащий только отрицательные элементы.

45. Вариант
Написать функцию находящую сумму элементов каждой диагонали матрицы, параллельной главной (начиная с одноэлементной диагонали $A_{0,m-1}$).

46. Вариант
Написать функцию зеркально отражающую элементы квадратной матрицы относительно побочной диагонали. Вспомогательную матрицу не использовать.

47. Вариант
Написать функцию поворачивающую матрицу на угол 180° (при этом элемент $A_{0,0}$ поменяется местами с $A_{M-1,M-1}$ и т. д.). Вспомогательную матрицу не использовать.

48. Вариант Дана матрица вещественных чисел. В каждом столбце найти максимальный и минимальный элементы, и сумму элементов, заключенных между ними.

49. Вариант
Написать функцию, которая в каждой строке матрицы подсчитывает количество элементов, предшествующих максимуму.

50. Вариант
Написать функцию, которая сортирует методом пузырька строки матрицы, все элементы которых четные.

51. Вариант
Написать функцию находящую максимальное из чисел, встречающихся в массиве более одного раза.

52. Вариант
Написать функцию обнуляющую элементы матрицы, лежащие одновременно выше главной и выше побочной диагоналей. Условный оператор внутри циклов не использовать (индексы внутреннего цикла начинать и заканчивать с учетом индексов внешнего цикла).

53. Вариант
Написать функцию которая заменяет каждый элемент массива, кроме начального и конечного, на его среднее арифметическое с предыдущим и последующим элементом (выполняет скользящее усреднение).

54. Вариант
Написать функцию, которая в массиве находит два числа, сумма которых наибольшая и делится нацело на 12.

55. Вариант
Написать функцию возвращающую массив из элементов исходного массива, которые встречается более одного раза.

56. Вариант
Написать функцию меняющая местами строки матрицы, содержащие минимальный и максимальный элементы.

57. Вариант

Написать функцию, которая заменяет значение каждого не положительного элемента массива X(n) абсолютной величиной соответствующего по номеру элемента массива Y(n).

58. Вариант

Написать функцию, которая формирует массив из элементов исходного массива, удовлетворяющего двойному неравенству $A_0 < A_K < A_{10}$.

59. Вариант

Упорядочить массив целых положительных чисел по возрастанию методом выбора и определить номер минимального простого числа

60. Вариант

Написать функцию, заменяющую числа массива X на соответствующие числа массива Y, если разница между ними больше заданного значения.