**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по курсовой работе**

**по дисциплине «Программирование»**

Тема: Основы алгоритмизации и программирования на языке С++

|  |  |  |
| --- | --- | --- |
| Студентка гр. 4372 |  | Хальметова Ю.Р. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Объединить 4 лабораторные работы в единый проект. Нужно добавить инфраструктуру переключения между заданиями (интерактивное меню).

1)Разработать алгоритм и написать программу, которая позволяет выводить сколько памяти отводится под различные типы данных, вывести на экран двоичное представление в памяти (все разряды) целого числа, вывести на экран двоичное представление в памяти (все разряды) типа float, вывести на экран двоичное представление в памяти (все разряды) типа double.

2)Разработать алгоритм и написать программу, которая позволяет создать целочисленный массив размером N = 100, заполнив его случайными значениями в диапазоне от -99 до 99. Затем массив нужно отсортировать методом сортировки от меньшего к большему, при этом важно измерить время, затраченное на сортировку, с использованием библиотеки chrono. После этого нужно найти максимальный и минимальный элементы массива, подсчитать время их поиска как в отсортированном, так и в неотсортированном массиве. Далее следует вычислить среднее значение максимального и минимального элементов, вывести индексы всех элементов, равных этим значениям, и их количество, также подсчитав время поиска. Затем нужно определить, сколько элементов в отсортированном массиве меньше заданного пользователем числа a и сколько элементов больше числа b, также введенного пользователем. Кроме того, необходимо проверить наличие введенного пользователем числа в отсортированном массиве с помощью алгоритма бинарного поиска и сравнить его скорость работы с обычным перебором. В завершение, следует реализовать обмен местами элементов массива по введенным пользователем индексам и вывести время, затраченное на этот обмен, используя библиотеку chrono.

3)Используя арифметику указателей, необходимо заполнить квадратичную целочисленную матрицу, после чего переставить блоки этой матрицы в соответствии с заданными схемами. Далее, применяя арифметику указателей, следует отсортировать одномерный массив. В завершение, необходимо провести операции уменьшения, увеличения, умножения или деления всех элементов матрицы на число, введенное пользователем.

4)Цель работы заключается в разработке программы, которая обрабатывает последовательность слов, введённых пользователем. Программа должна реализовывать следующие функции: удаление лишних пробелов между словами, устранение повторяющихся знаков препинания, корректировка регистра букв, а также добавление точки в конце текста, если это необходимо. Ввод данных может осуществляться как с клавиатуры, так и из файла, что предоставляет пользователю гибкость в выборе способа ввода.

**Основные теоретические положения.**

1. Внутреннее представление величин целого типа – целое число в двоичном коде. При использовании спецификатора signed старший бит числа интерпретируется как знаковый (0 – положительное число, 1 – отрицательное). Для кодирования целых чисел со знаком применяется прямой, обратный и дополнительный коды.

Представление положительных и отрицательных чисел в прямом, обратном и дополнительном кодах отличается. В прямом коде в знаковый разряд помещается цифра 1, а в разряды цифровой части числа – двоичный код его абсолютной величины. Прямой код числа −3 (для 16- разрядного процессора):



Обратный код получается инвертированием всех цифр двоичного кода абсолютной величины, включая разряд знака: нули заменяются единицами, единицы – нулями. Прямой код можно преобразовать в обратный, инвертировав все значения всех битов (кроме знакового). Обратный код числа −3:



Дополнительный код получается образованием обратного кода с последующим прибавлением единицы к его младшему разряду. Дополнительный код числа −3:



Увидеть, каким образом тип данных представляется на компьютере, можно при помощи логических операций: побитового сдвига (<<) и поразрядной конъюнкции (&).

putchar(value & mask ? '1' : '0'); // если 1, то возвращается 1, иначе 0

value <<= 1; // побитовый сдвиг влево на 1 бит

Putchar возвращает один символ в консоль. Альтернатива - cout. В представленном способе, маска - то, с чем сравнивается значение. И побитовый сдвиг применяется для value. Таким образом 1 бит будет сравниваться с каждым битом числа. Альтернатива - побитовый сдвиг вправо, но при этом нужно проводить данную операцию не над значением(единицей), а над маской (исходым числом, битовое представление которого нужно получить).

При сдвиге вправо для чисел без знака позиции битов, освобожденные при операции сдвига, заполняются нулями. Для чисел со знаком бит знака используется для заполнения освобожденных позиций битов. Другими словами, если число 25 является положительным, используется 0, если число является отрицательным, используется 1. При сдвиге влево позиции битов, освобожденных при операции сдвига, заполняются нулями. Сдвиг влево является логическим сдвигом (биты, сдвигаемые с конца, отбрасываются, включая бит знака).

Вещественные типы данных хранятся в памяти компьютера иначе, чем целочисленные. Внутреннее представление вещественного числа состоит из двух частей – мантиссы и порядка.

Для 32-разрядного процессора для float под мантиссу отводится 23 бита, под экспоненту – 8, под знак – 1. Для double под мантиссу отводится 52 бита, под экспоненту – 11, под знак – 1:



Увидеть, каким образом вещественные типы данных представляются в компьютере немного сложнее. Логические операции, которые использовались с int, для вещественных типов данных не подходят. Но это ограничение можно легко обойти, использовав объединения.

Объединения – это две или более переменных расположенных по одному адресу (они разделяют одну и ту же память). Объединения определяются с использованием ключевого слова union. Объединения не могут хранить одновременно несколько различных значений, они позволяют интерпретировать несколькими различными способами содержимое одной и той же области памяти.

С объединениями нужно быть острожным. Вся работа с памятью требует грамотного подхода. Более подробно с объединениями можно будет ознакомиться при изучении структур. Пока что объедения будут служить инструментом для работы с float и double.

#include <iostream>

using namespace std;

int main()

{

union {

int tool;

float numb\_f = 3.14;

};

cout << tool << endl; // 1078523331

cout << numb\_f << endl; // 3.14

tool = tool >> 1; // побитовый сдвиг вправо

cout << tool << endl; // 5392261665

cout << numb\_f; // 1.3932e-19

return 0;

}

Подобные манипуляции возможны благодаря тому, что int и float занимают 4 байта. Проводя манипуляции над tool, мы изменяем значение numb\_f. Таким образом, алгоритм, который использовался для представления в памяти int может использоваться и для float.

Алгоритма представления double немного отличается. Под вещественное число с двойной точностью отводиться 8 байт, в то время как под int всего 4 байта. Но и это ограничение можно легко обойти. Так как данные любой линейной структуры в память записываются последовательно (друг за другом), можно использовать массив из двух int, под который будет отведено 8 байт.

#include <iostream>

using namespace std;

int main()

{

int value = -127; // Значение числа

unsigned int order = 32; // Количество разрядов

unsigned int mask = 1 << order – 1; // Маска побитового сравнения

for (int i = 1; i <= order; i++)

{

putchar(value & mask ? '1' : '0');

value <<= 1; // Побитовый сдвиг числа

if (i % 8 == 0)

{

putchar(' ');

}

if (i % order – 1 == 0)

{

putchar(' ');

}

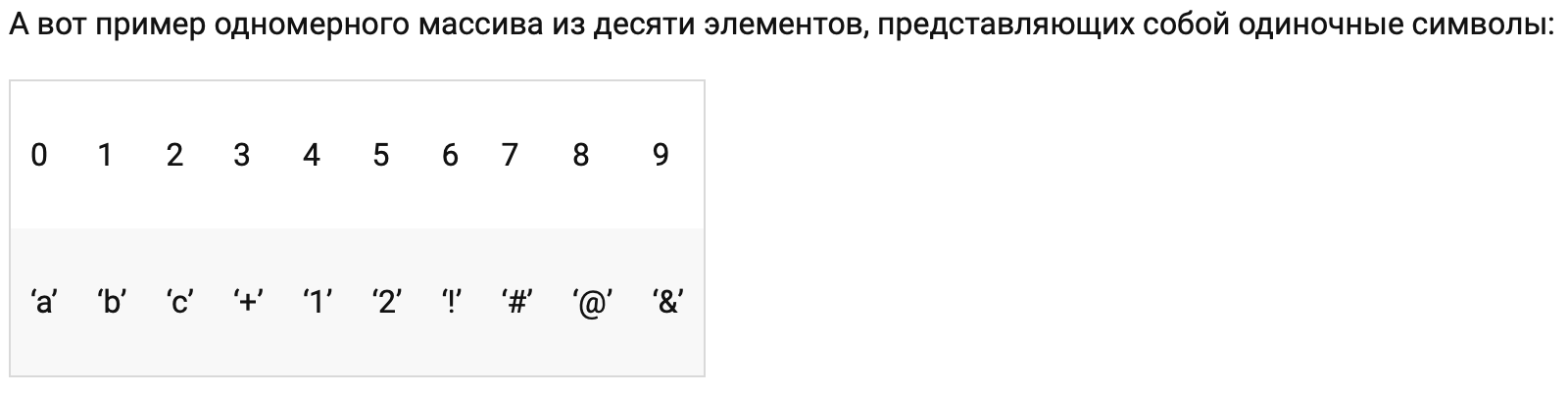
}

return 0;

}

В консоль будет выведено: 1 1111111 11111111 11111111 10000001.

2) Массив представляет собой индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение. Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных.



Объявление в программах одномерных массивов выполняется в соответствии со следующим правилом:

<Базовый тип элементов> <Идентификатор массива> [<Количество элементов>]

Все массивы можно разделить на две группы: одномерные и многомерные. Описание массива в программе отличается от объявления обычной переменной наличием размерности массива, которая задается в квадратных скобках после имени.

Многомерные массивы определяются аналогично одномерным массивам. Количество элементов по каждому измерению указывается отдельно в квадратных скобках:

int  A1 [5] [3];                 //  Двумерный массив, элементами которого являются

                                         //   значения типа int

double D [10] [15] [3];   //  Трехмерный массив, элементами которого являются

                                 //   значения типа double

Простейший циклический алгоритм вывода значений элементов некоторого одномерного массива выглядит так:

const int n = 10;

short A[n];

//  Для использования setw() необходимо включить #include <iomanip>

for (int i = 0; i < n; ++i)

       cout << setw(8) << left << A[i];

cout << endl;

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Когда элементы отсортированы, их проще найти, производить с ними различные операции. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива.

Обмен элементов массива осуществляется через буферную переменную либо через функцию swap(a, b).

Простейшая из сортировок – сортировка обменом (пузырьковая сортировка). Вся суть метода заключается в попарном сравнении элементов и последующем обмене. Таким образом, если следующий элемент меньше текущего, то они меняются местами, максимальный элемент массива постепенно смещается в конец массива, а минимальный – в начало. Один полный проход по массиву может гарантировать, что в конце массива находится максимальный элемент.

Shaker sort – модификация пузырьковой сортировки. Принцип работы этой сортировки аналогичен bubble sort: попарное сравнение элементов и последующий обмен местами. Но имеется существенное отличие. Как только максимальный элемент становится на свое место, алгоритм не начинает новую итерацию с первого элемента, а запускает сортировку в обратную сторону. Алгоритм гарантирует, что после выполнения первой итерации, минимальный и максимальный элемент будут в начале и конце массива соответственно.

Comb sort (сортировка расческой) – ещё одна модификация сортировки пузырьком. Алгоритм был разработан специально для случаев, когда минимальные элементы стоят слишком далеко, или максимальные – слишком близко к началу массива. В сортировке расческой переставляются элементы, стоящие на расстоянии.

Сортировка вставками (insert sort) – алгоритм сортировки, в котором элементы массива просматриваются по одному, и каждый новый элемент размещается в подходящее место среди ранее упорядоченных элементов.

Быстрая сортировка (quick sort) – одна из самых быстрых сортировок. Эта сортировка по сути является существенно улучшенной версией алгоритма пузырьковой сортировки. Общая идея алгоритма состоит в том, что сначала выбирается из массива элемент, который называется опорным. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Затем необходимо сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: меньше опорного, раны опорному и больше опорного. Для меньших и больших значений необходимо выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

Алгоритм бинарного поиска – классический алгоритм поиска в отсортированном массиве, который использует дробление массива на половины. Если элемент, который необходимо найти, присутствует в списке, то бинарный поиск возвращает ту позицию, в которой он был найден.

3) Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения. Указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя.

Присвоение указателю адреса уже рассматривалось в прошлой теме. Для получения адреса объекта используется операция &:

int a = 1917;

int \*pa = &a;   // указатель pa хранит адрес переменной a

При этом указатель и переменная должны иметь один и тот же тип, в данном случае это тип int.

Присвоение указателю другого указателя:

#include <iostream>

using std::cout;

using std::endl;

int main()

{

int a = 10;

int b = 2;

int \*pa = &a;

int \*pb = &b;

cout << "Variable a: address=" << pa << "\t value=" << \*pa << endl;

cout << "Variable b: address=" << pb << "\t value=" << \*pb << endl;

pa = pb;    // теперь указатель pa хранит адрес переменной b

cout << "Variable b: address=" << pa << "\t value=" << \*pa << endl;

return 0;

}

Когда указателю присваивается другой указатель, то фактически первый указатель начинает также указывать на тот же адрес, на который указывает второй указатель.

**Операция разыменования**указателя представляет выражение в виде \*имя\_указателя. Эта операция позволяет получить объект по адресу, который хранится в указателе.

#include <iostream>

using std::cout;

using std::endl;

int main()

{

   int a = 10;

   int \*pa = &a;

   int \*pb = pa;

   \*pa = 25;

   cout << "Value on pointer pa: " << \*pa << endl;  // 25

   cout << "Value on pointer pb: " << \*pb << endl;  // 25

   cout << "Value of variable a: " << a << endl;    // 25

   return 0;

}

Через выражение \*pa мы можем получить значение по адресу, который хранится в указателе pa, а через выражение типа \*pa = значение вложить по этому адресу новое значение.

И так как в данном случае указатель pa указывает на переменную a, то при изменении значения по адресу, на который указывает указатель, также изменится и значение переменной a.

Указателю, имеющему такой же базовый тип, как и элементы массива, можно присвоить массив следующим образом:

int Arr[10];

int \*p;

p = Arr;

4) Текстовые строки представляются с помощью одномерных массивов символов. В языке C++ текстовая строка представляет собой набор символов, обязательно заканчивающийся нулевым символом (‘\0’).

Класс string предназначен для работы со строками типа char, которые представляют собой строчку с завершающим нулем (символ ‘\0’). Класс string был введен как альтернативный вариант для работы со строками типа char.

При работе со строками часто будет возникать потребность в поиске набора символа или слов (поиска подстроки в строке). При условии, что текст может быть крайне большим, хочется, чтобы алгоритм поиска подстроки работал быстро.

Самый простой способ подстроки в строке – Линейный поиск – циклическое сравнение всех символов строки с подстрокой. Действительно, этот способ первый приходит в голову, но очевидно, что он будет самым долгим.

**Постановка задачи.**

1. Необходимо разработать алгоритм и написать программу, которая позволяет выводить, сколько памяти (в байтах) на компьютере отводится под различные типы данных со спецификаторами и без: int, short int, long int, float, double, long double, char и bool,  выводить на экран двоичное представление в памяти (все разряды) целого числа, выводить на экран двоичное представление в памяти (все разряды) типа float, выводить на экран двоичное представление в памяти (все разряды) типа double.
2. 1. Создать целочисленный массив размером N = 100, заполнить его случайными значениями в диапазоне от -99 до 99.

2. Отсортировать массив методом сортировки от меньшего к большему, измеряя время, затраченное на сортировку, с использованием библиотеки chrono.

3. Найти максимальный и минимальный элементы массива, подсчитывая время их поиска как в отсортированном, так и в неотсортированном массиве.

4. Вычислить среднее значение максимального и минимального элементов, вывести индексы всех элементов, равных этим значениям, и их количество, также подсчитывая время поиска.

5. Определить количество элементов в отсортированном массиве, которые меньше заданного пользователем числа a, и количество элементов, которые больше числа b, также введённого пользователем.

6. Проверить наличие введённого пользователем числа в отсортированном массиве с помощью алгоритма бинарного поиска, сравнить его скорость работы с обычным перебором.

7. Реализовать обмен местами элементов массива по введённым пользователем индексам и вывести время, затраченное на этот обмен, с использованием библиотеки chrono.

1. Используя арифметику указателей, заполнить квадратичную целочисленную матрицу; Переставить ее блоки в соответствии со схемами; Используя арифметику указателей сортировать массив; Уменьшить, увеличить, умножить или делить все элементы матрицы на введенное пользователем число.
2. Постановка задачи заключается в разработке программы, которая будет обрабатывать последовательность слов, вводимых пользователем. Программа должна выполнять следующие функции: удалять избыточные пробелы между словами, устранять повторяющиеся знаки препинания, корректировать регистр букв и добавлять точку в конце текста, если это необходимо. Ввод данных должен осуществляться как с клавиатуры, так и из файла, что обеспечит пользователю удобство и гибкость в выборе способа ввода.

**Выполнение работы.**

| При запуске программы перед пользователем появляется, выбор задания. |  |
| --- | --- |
| Меню | |
| При запуске программы перед пользователем появляется, выбор задания. | Меню: |
| Вывод количества памяти (в байтах), отводимого под различные типы данных со спецификаторами и без. | |
| При вводе пользователем корректного значения пункта меню программа выполняется.  Для выполнения использовалась конструкция sizeof(). | Для выполнения первого задания нужно вывести сколько памяти (в байтах) на компьютере отводится под различные типы данных со спецификаторами и без  Вывод на экран двоичного представления в памяти (все разряды) целого числа. |
| Пользователь вводит целое число и в результате получает его двоичное представление в памяти. | Для выполнения второго задания нужно вывести на экран двоичное представление в памяти (все разряды) целого числа |
| Вывод на экран двоичного представления в памяти (все разряды) типа float. | |
| Пользователь вводит вещественное число, а на выходе получает его двоичное представление в памяти | Для выполнения третьего задания нужно вывести на экран двоичное представление в памяти (все разряды) типа float |
| Вывод на экран двоичного представления в памяти (все разряды) типа double. | |
| Пользователь вводит вещественное число, а на выходе получает его двоичное представление в памяти. | Для выполнения третьего задания нужно вывести на экран двоичное представление в памяти (все разряды) типа double. |
| Меню | | |
| При запуске программы перед пользователем появляется, выбор задания. | Меню: | |
| Вывод целочисленного массива размерности 100 . | | |
| При вводе пользователем корректного значения пункта меню программа выполняется. | Создает целочисленный массив размерности N = 100. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99. | |
| Вывод на экран созданного отсортированного массива. | | |
| При вводе пользователем корректного значения пункта меню программа выполняется. | Сортирует массив и выводит на экран время сортировки в наносекундах | |
| Вывод на экран минимального и максимального элемента массива с замером времени | | |
| При вводе пользователем корректного значения пункта выполняется поиск наименьшего и наибольшего элементов массива | Ищет наибольший и наименьший элементы массива, замеряет время поиска и выводит на экран | |
| Вывод на экран среднего значения (если необходимо, число нужно округлить) максимального и минимального значения в отсортированном и неотсортированном. Вывод индексов всех элементов, которые равны этому значению, и их количество | | |
| При вводе пользователем корректного значения подсчет среднего значения максимального и минимального элементов | Считает среднее значение макс. и мин. элементов и выводит на экран индексы всех элементов, которые равны этому значению. Замеряет время подсчета и поиска и выводит на экран время в наносекундах. | |
| Вывод на экран количества элементов в отсортированном массиве, которые меньше числа a, которое инициализируется пользователем. | | |
| При вводе пользователем корректного значения пользователь вводит число a. | Производит подсчет чисел в массиве, которые меньше введенного пользователем числа, вывод на экран | |
| Вывод на экран количества элементов в отсортированном массиве, которые больше числа b, которое инициализируется пользователем. | | |
| При вводе пользователем корректного значения пользователь вводит число b. | Производит подсчет чисел в массиве, которые больше введенного пользователем числа, вывод на экран | |
| Вывод на экран информации о том, есть ли введенное пользователем число в отсортированном массиве. | | |
| При вводе пользователем корректного значения пользователь вводит число для поиска его в массиве. | Производит поиск введенного пользователем числа при помощи бинарного поиска и поиска перебором. Замеряет время подсчета и поиска и выводит на экран время в наносекундах. | |
| Обмен местами элементов массива, индексы которых вводит пользователь | | |
| При вводе пользователем корректного значения пользователь вводит число. | Производит обмен местами элементов, индексы которых вводит пользователь. Производит подсчет времени и выводит на экран. | |
| Меню | | |
| При запуске программы перед пользователем появляется, выбор задания. | Меню: | |
| Вывод квадратичной целочисленной матрицы порядка N (6,8,10) случайными числами от 1 до  N\*N | | |
| При вводе пользователем корректного значения пункта меню пользователь выбирает способ вывода матрицы | Заполняет квадратичную матрицу и выводит ее, исходя из выбора пользователя | |
| Вывод на экран созданного массива после перестановки блоков . | | |
| При вводе пользователем корректного значения пункта меню пользователь выбирает способ перестановки матрицы | Переставляет блоки квадратичной матрицы и выводит ее, исходя из выбора пользователя | |
| Сортировка массива | | |
| При запуске программы перед пользователем появляется, выбор задания. | Сортирует матрицу и выводит ее на экран | |
| Над каждым элементом матрицы происходит действие исходя из выбора пользователя | | |
| Пользователь выбирает действие, которое будет производиться над каждым элементом матрицы | Изменяет каждый элемент матрицы и выводит ее на экран | |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| Меню |
| При запуске программы перед пользователем появляется, выбор задания. | Меню: |
| С клавиатуры или с файла (\*) (пользователь сам может выбрать способ ввода) вводится последовательность, содержащая от 1 до 50 слов, в каждом из которых от 1 до 10 строчных латинских букв и цифр. Между соседними словами произвольное количество пробелов. За последним символом стоит точка. |
| При вводе пользователем корректного значения пункта меню пользователь выбирает способ ввода текста | Запрашивает у пользователя способ ввода текста и выводит слова на экран |
| Редактирует входной текст |
| При запуске программы перед пользователем появляется, выбор задания. | Удаляет лишние пробелы, лишние знаки препинания и исправляет регистр букв во входном тексте и выводит отредактированный текст на экран |
| Удаление из массива заданного набора букв или цифр |
| При запуске программы пользователь вводит набор символов, который будет удален из текста. | Удаляет из массива заданный набор символов и выводит на экран измененный текст | |
| Поиск всех подстрок, которую введет пользователь |
| Пользователь задает подстроку для поиска ее в тексте | Производит поиск при помощи алгоритма Бойера-Мура и выводит все подстроки, которую введет пользователь | |

**Выводы.**

1)В процессе выполнения работы были изучены типы данных с и без спецификаторов, а также разряды чисел. Освоены различные логические операции, такие как побитовый сдвиг (<<) и поразрядная конъюнкция (&). Также была разработана программа, позволяющая определять объем памяти (в байтах), выделяемый под различные типы данных, а также выводить на экран двоичное представление целого числа и типов float и double в памяти.

2) В процессе выполнения работы были изучены одномереные статические массивы, библиотека сhrono и типы сортировок.

3) Используя арифметику указателей, заполнить квадратичную целочисленную матрицу; Переставить ее блоки в соответствии со схемами; Используя арифметику указателей сортировать массив; Уменьшить, увеличить, умножить или делить все элементы матрицы на введенное пользователем число.

4) Выводом работы является разработанная программа, которая обрабатывает последовательность слов, введённых пользователем. Программа успешно реализует следующие функции: удаление лишних пробелов между словами, устранение повторяющихся знаков препинания, корректировка регистра букв и добавление точки в конце текста, если это необходимо. Пользователь имеет возможность вводить данные как с клавиатуры, так и из файла, что обеспечивает гибкость в выборе способа ввода.

Приложение А

рабочий код

#include <iostream>

#include <unistd.h>

#include <unordered\_set>

#include <vector>

#include <chrono>

#include <cstdlib>

#include <set>

#include <iomanip>

#include <string>

#include <cctype>

#include <regex>

#include <fstream>

#include <sstream>

using namespace std;

using namespace chrono;

const int rows = 6;

const int cols = 6;

void idz\_32(int num)

{

int number\_bit;

cout << "Введите номер бита: ";

cin >> number\_bit;

num &= ~(1 << number\_bit);

cout << "Число после применения идз: ";

for (int i = sizeof(int) \* 8 - 1; i >=0; i --)

{

cout << ((num & (1 << i)) ? "1" : "0");

if (i == 31 || i % 8 == 0)

{

cout << " ";

}

}

cout << '\n';

cout << "\n";

cout << "Конечный результат: "<< num << endl;

}

int binarySearch(int arr[], float number, int start, int end) {

if (end >= start) {

int mid = start + (end - start) / 2;

if (arr[mid] == number) {

return mid;

}

if (arr[mid] > number) {

return binarySearch(arr, number, start, mid - 1);

} else {

return binarySearch(arr, number, mid + 1, end);

}

}

return -1;

}

void quickSort(int arr[],int n ,int end, int start){

int mid;

int f = start;

int l = end;

mid = arr[(f+l)/2];

while (f < l){

while (arr[f] < mid) f++;

while (arr[l] > mid) l--;

if (f <= l) {

swap(arr[f], arr[l]);

f++;

l--;

}

}

}

void insertionSort(int arr[], int n){

int i, j, key;

for (i = 1; i < n; i++){

key = arr[i];

j = i - 1;

while (j >=0 && arr[j] > key){

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

void combSort(int arr[], int n){

float k = 1.247;

int s;

s = n-1;

while (s >= 1){

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

if (arr[i] > arr[i + s]){

swap(arr[i], arr[i + s]);

}

}

s/=k;

}

}

void cocktailSort(int arr[], int n){

bool swapped = true;

int start = 0;

int end = n-1;

while (swapped){

swapped = false;

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

if (arr[i] > arr[i + 1]){

swap(arr[i], arr[i + 1]);

swapped = true;

}

}

if (!swapped){

break;

}

end--;

for(int i = end - 1;i >= start; i--){

if (arr[i] > arr[i + 1]){

swap(arr[i], arr[i + 1]);

swapped = true;

}

}

start++;

}

}

void bubbleSort(int arr[], int n) {

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

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

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

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

}

}

}

}

void intToFloatMatrix(const int\* intMatrix, float\* floatMatrix, size\_t rows, size\_t cols) {

for (size\_t i = 0; i < rows; ++i) {

for (size\_t j = 0; j < cols; ++j) {

\*(floatMatrix + i \* cols + j) = static\_cast<float>(\*(intMatrix + i \* cols + j));

}

}

}

void plusNumber(int\* arr, int rows, int cols, int n) {

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

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

\*(arr + i \* cols + j) += n;

}

}

}

void minusNumber(int\* arr, int rows, int cols, int n) {

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

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

\*(arr + i \* cols + j) -= n;

}

}

}

void multiplyNumber(int\* arr, int rows, int cols, int n) {

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

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

\*(arr + i \* cols + j) \*= n;

}

}

}

void shareNumber(float\* f\_arr, int rows, int cols, int n) {

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

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

\*(f\_arr + i \* cols + j) /= n;

}

}

cout << "Измененная матрица: " << endl;

cout << fixed << setprecision(1);

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

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

cout << left << setw(10) << \*(f\_arr + i \* cols + j);

usleep(50000);

}

cout << endl;

}

}

void fillSpiral(int\* arr, int rows, int cols) {

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

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

\*(arr + i \* cols + j) = 1 + rand() % (rows \* cols);

}

}

}

void straightArray(int(\*arr)[cols], int\* flatArray) {

int index = 0;

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

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

\*(flatArray + index++) = \*(\*(arr + i) + j);

}

}

}

void swap(int\* a, int\* b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void quickSortIndicator(int\* arr, int start, int end) {

if (start >= end) return;

int f = start;

int l = end;

int mid = \*(arr + (f + l) / 2);

while (f <= l) {

while (\*(arr + f) < mid) f++;

while (\*(arr + l) > mid) l--;

if (f <= l) {

swap(arr + f, arr + l);

f++;

l--;

}

}

quickSortIndicator(arr, start, l);

quickSortIndicator(arr, f, end);

}

void insertionSortIndicator(int\* arr, int n) {

int i, j, key;

for (i = 1; i < n; i++) {

key = \*(arr + i);

j = i - 1;

while (j >= 0 && \*(arr + j) > key) {

\*(arr + j + 1) = \*(arr + j);

j--;

}

\*(arr + j + 1) = key;

}

}

void combSortIndicator(int\* arr, int n) {

float k = 1.247;

int s = n - 1;

while (s >= 1) {

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

if (\*(arr + i) > \*(arr + i + s)) {

swap(\*(arr + i), \*(arr + i + s));

}

}

s /= k;

}

}

void cocktailSortIndicator(int\* arr, int n) {

bool swapped = true;

int start = 0;

int end = n - 1;

while (swapped) {

swapped = false;

for (int i = start; i < end; i++) {

if (\*(arr + i) > \*(arr + i + 1)) {

swap(\*(arr + i), \*(arr + i + 1));

swapped = true;

}

}

if (!swapped) {

break;

}

end--;

for (int i = end; i >= start; i--) {

if (\*(arr + i) > \*(arr + i + 1)) {

swap(\*(arr + i), \*(arr + i + 1));

swapped = true;

}

}

start++;

}

}

void bubbleSortIndicator(int\* arr, int n) {

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

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

if (\*(arr + j) > \*(arr + j + 1)) {

swap(\*(arr + j), \*(arr + j + 1));

}

}

}

}

void fillSpiralSorted(int (\*arr)[cols]) {

int flatArray[rows \* cols];

straightArray(arr, flatArray);

quickSortIndicator(flatArray,0, rows \* cols - 1);

int index = 0;

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

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

\*(\*(arr + i) + j) = \*(flatArray + index++);

}

}

}

void setCursorPosition(int x, int y) {

cout << "\033[" << y << ";" << x << "H";

}

void printMatrix(int arr[][cols], int rows, int cols) {

int\* ptr = (int\*)arr;

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

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

cout << \*(ptr + i \* cols + j) << " ";

usleep(50000);

}

cout << endl;

}

}

void printSpiral(int arr[][cols], int rows, int cols) {

system("clear");

int top = 0, bottom = rows - 1;

int left = 0, right = cols - 1;

int\* ptr = (int\*)arr;

while (top <= bottom && left <= right) {

for (int i = left; i <= right; i++) {

setCursorPosition(i \* 4, top + 1);

cout << \*(ptr + top \* cols + i);

cout.flush();

usleep(300000);

}

top++;

for (int i = top; i <= bottom; i++) {

setCursorPosition(right \* 4, i + 1);

cout << \*(ptr + i \* cols + right);

cout.flush();

usleep(300000);

}

right--;

if (top <= bottom) {

for (int i = right; i >= left; i--) {

setCursorPosition(i \* 4, bottom + 1);

cout << \*(ptr + bottom \* cols + i);

cout.flush();

usleep(300000);

}

bottom--;

}

if (left <= right) {

for (int i = bottom; i >= top; i--) {

setCursorPosition(left \* 4, i + 1);

cout << \*(ptr + i \* cols + left);

cout.flush();

usleep(300000);

}

left++;

}

}

setCursorPosition(0, rows + 2);

}

void printSnake(int arr[][cols], int rows, int cols) {

system("clear");

int\* ptr = (int\*)arr;

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

if (j % 2 == 0) {

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

setCursorPosition(j \* 4, i + 1);

cout << \*(ptr + i \* cols + j);

cout.flush();

usleep(150000);

}

} else {

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

setCursorPosition(j \* 4, i + 1);

cout << \*(ptr + i \* cols + j);

cout.flush();

usleep(300000);

}

}

}

setCursorPosition(0, rows + 2);

}

void change\_A(int (\*arr)[cols])

{

int blockSize = rows / 2;

int temp[rows][cols];

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

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

\*(\*(temp + i) + (j + blockSize)) = \*(\*(arr + i) + j);

\*(\*(temp + (i + blockSize)) + (j + blockSize)) = \*(\*(arr + i) + (j + blockSize));

\*(\*(temp + (i + blockSize)) + j) = \*(\*(arr + (i + blockSize)) + (j + blockSize));

\*(\*(temp + i) + j) = \*(\*(arr + (i + blockSize)) + j);

}

}

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

{

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

{

\*(\*(arr + i) + j) = \*(\*(temp + i) + j);

}

}

}

void change\_B(int (\*arr)[cols]) {

int blockSize = rows / 2;

int new\_arr[rows][cols];

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

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

\*(\*(new\_arr + (i + blockSize)) + (j + blockSize)) = \*(\*(arr + i) + j);

\*(\*(new\_arr + i) + j) = \*(\*(arr + (i + blockSize)) + (j + blockSize));

\*(\*(new\_arr + (i + blockSize)) + j) = \*(\*(arr + i) + (j + blockSize));

\*(\*(new\_arr + i) + (j + blockSize)) = \*(\*(arr + (i + blockSize)) + j);

}

}

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

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

\*(\*(arr + i) + j) = \*(\*(new\_arr + i) + j);

}

}

}

void change\_C(int (\*arr)[cols]) {

int blockSize = rows / 2;

int new\_arr[rows][cols];

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

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

\*(\*(new\_arr + (i + blockSize)) + j) = \*(\*(arr + i) + j);

\*(\*(new\_arr + i) + j) = \*(\*(arr + (i + blockSize)) + j);

\*(\*(new\_arr + (i + blockSize)) + (j + blockSize)) = \*(\*(arr + i) + (j + blockSize));

\*(\*(new\_arr + i) + (j + blockSize)) = \*(\*(arr + (i + blockSize)) + (j + blockSize));

}

}

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

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

\*(\*(arr + i) + j) = \*(\*(new\_arr + i) + j);

}

}

}

void change\_D(int (\*arr)[cols]) {

int blockSize = rows / 2;

int new\_arr[rows][cols];

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

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

\*(\*(new\_arr + i) + (j + blockSize)) = \*(\*(arr + i) + j);

\*(\*(new\_arr + i) + j) = \*(\*(arr + i) + (j + blockSize));

\*(\*(new\_arr + (i + blockSize)) + (j + blockSize)) = \*(\*(arr + (i + blockSize)) + j);

\*(\*(new\_arr + (i + blockSize)) + j) = \*(\*(arr + (i + blockSize)) + (j + blockSize));

}

}

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

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

\*(\*(arr + i) + j) = \*(\*(new\_arr + i) + j);

}

}

}

unordered\_map<char, int> createBadCharTable(const string& pattern) {

unordered\_map<char, int> badCharTable;

int length = pattern.length();

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

badCharTable[pattern[i]] = i;

}

return badCharTable;

}

vector<int> boyerMooreSearch(const string& text, const string& pattern) {

vector<int> positions;

int textLength = text.length();

int patternLength = pattern.length();

if (patternLength == 0 || textLength < patternLength) {

return positions;

}

auto badCharTable = createBadCharTable(pattern);

int shift = 0;

while (shift <= textLength - patternLength) {

int j = patternLength - 1;

while (j >= 0 && pattern[j] == text[shift + j]) {

j--;

}

if (j < 0) {

positions.push\_back(shift);

shift += (shift + patternLength < textLength) ?

patternLength - badCharTable[text[shift + patternLength]] : 1;

} else {

shift += max(1, j - badCharTable[text[shift + j]]);

}

}

return positions;

}

string removeCharacters(const string& word, const string& charsToRemove) {

string result;

for (char ch : word) {

if (charsToRemove.find(ch) == string::npos) {

result += ch;

}

}

return result;

}

bool containsDigits(const string& word) {

for (char ch : word) {

if (isdigit(ch)) {

return true;

}

}

return false;

}

string toLowerCase(const string& input) {

stringstream ss(input);

string word;

string result;

while (ss >> word) {

for (char& ch : word) {

ch = tolower(ch);

}

if (!result.empty()) {

result += " ";

}

result += word;

}

return result;

}

string removeExtraPunctuation(const string& input) {

stringstream ss(input);

string result;

char lastChar = '0';

int dotCount = 0;

string punctuation = "!#$%&'()\*+,-./:;<=>?@[^\_{|}~";

for (char currentChar : input) {

if (punctuation.find(currentChar) != string::npos) {

if (currentChar == '.') {

dotCount++;

if (dotCount <= 3) {

result += currentChar;

}

continue;

} else {

dotCount = 0;

if (currentChar == lastChar) {

continue;

}

}

} else {

dotCount = 0;

}

result += currentChar;

lastChar = currentChar;

}

return result;

}

string removeExtraSpaces(const string& input) {

stringstream ss(input);

string word;

string result;

while (ss >> word) {

if (!result.empty()) {

result += " ";

}

result += word;

}

return result;

}

bool isValidInput(const string& input, vector<string>& words) {

if (input.empty() || input.back() != '.') {

cout << "Ввод должен заканчиваться точкой." << endl;

return false;

}

string trimmedInput = input.substr(0, input.size() - 1);

istringstream iss(trimmedInput);

string word;

while (iss >> word) {

if (word.length() < 1 || word.length() > 10) {

cout << "Каждое слово должно содержать от 1 до 10 символов." << endl;

return false;

}

words.push\_back(word);

}

if (words.size() < 1 || words.size() > 50) {

cout << "Количество слов должно быть от 1 до 50." << endl;

return false;

}

return true;

}

string inputFromKeyboard() {

string userInput;

cout << "Введите последовательность слов (завершите точкой): ";

getline(cin, userInput);

return userInput;

}

string inputFromFile() {

string filename;

cout << "Введите имя файла: ";

cin >> filename;

ifstream file(filename);

if (!file) {

cout << "Файл не найден. Попробуйте снова." << endl;

return inputFromFile();

}

string content;

string line;

while (getline(file, line)) {

content += line;

}

return content;

}

int main(){

system("clear");

bool flagPractical = true;

while (flagPractical){

cout << "Введите номер практической работы (1-4): ";

int taskPractical;

cin >> taskPractical;

switch(taskPractical){

case 1:{

unsigned int mask = 1 << (sizeof(int) \* 8 - 1);

int number, answer\_int, answer\_float, answer\_double;

bool flag\_1 = true;

while(flag\_1){

cout << "Введите номер задания(1-4, 5 - завершение программы): ";

cin >> number;

cout << "\n";

switch (number){

case 1:{

cout << "int: " << sizeof(int) << " байт\n";

cout << "short int: " << sizeof(short int) << " байт\n";

cout << "long int: " << sizeof(long int) << " байт\n";

cout << "float: " << sizeof(float) << " байт\n";

cout << "double: " << sizeof(double) << " байт\n";

cout << "long double: " << sizeof(long double) << " байт\n";

cout << "char: " << sizeof(char) << " байт\n";

cout << "bool: " << sizeof(bool) << " байт\n";

cout << "\n";

break;

}

case 2:{

cout << "Введите int значение: ";

int a;

cin >> a;

mask = 1 << (sizeof(int) \* 8 - 1);

for (int i = 1; i <= sizeof(int) \* 8; i++, mask >>= 1) {

if (i == 1) {

cout << "\033[31m";

cout << (a & mask ? "1" : "0");

cout << "\033[0m";

}

if (i != 1) {

if (a & mask) {

cout << "1";

}

else {

cout << "0";

}

}

if (i == 1 || i % 8 == 0) {

cout << " ";

}

}

cout << "\n";

cout << "\n";

cout << "Желаете применить идз к данному числу? (1 - да, 0 - нет): ";

cin >> answer\_int;

if (answer\_int == 1)

{

idz\_32(a);

}

else

{

break;

}

cout << '\n';

break;

}

case 3:{

union{

float b;

int ib;

};

cout << "Введите значение float: ";

cin >> b;

mask = 1 << (sizeof(float) \* 8 - 1);

for (int i = 1; i <= sizeof(float) \* 8; i++, mask >>= 1) {

if (i == 1) {

cout << "\033[35m";

cout << (ib & mask ? "1" : "0");

cout << "\033[0m";

}

if (i >=2 && i <= 8) {

cout << "\033[31m";

cout << (ib & mask ? "1" : "0");

cout << "\033[0m";

}

if (i > 8) {

if (ib & mask) {

cout << "1";

}

else {

cout << "0";

}

}

if (i == 1 || i % 8 == 0) {

cout << " ";

}

}

cout << "\n";

cout << "\n";

cout << "Желаете применить идз к данному числу? (1 - да, 0 - нет): ";

cin >> answer\_float;

if (answer\_float == 1)

{

idz\_32(ib);

}

else

{

break;

}

break;

}

case 4:{

union

{

double idouble;

int massiv[2];

};

cout << "Введите значение double: ";

cin >> idouble;

int count = 0;

string izm\_double;

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

unsigned int mask = 1 << (sizeof(float) \* 8 - 1);

for (int k = 1; k <= 32; k++,mask >>= 1){

count++;

if (count == 1)

{

cout << "\033[32m";

cout << (massiv[i] & mask ? "1" : "0");

cout << "\033[0m";

}

if (count > 1 && count <= 11)

{

cout << "\033[33m";

cout << (massiv[i] & mask ? "1" : "0");

cout << "\033[0m";

}

if (count > 11)

{

cout << "\033[35m";

cout << (massiv[i] & mask ? "1" : "0");

cout << "\033[0m";

}

if (count == 1 || count == 11)

{

cout << " ";

}

}

}

cout << "\n";

cout << "\n";

cout << "Желаете применить идз к данному числу? (1 - да, 0 - нет): ";

cin >> answer\_double;

if (answer\_double == 1)

{

{

if (idouble < 0)

{

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < sizeof(int) \* 8;j+=2)

{

massiv[i] &= ~(1 << j);

}

}

}

else

{

for (int i = 0; i < 2; i++)

{

for (int j = 1; j < sizeof(int) \* 8;j+=2)

{

massiv[i] |= (1 << j);

}

}

}

int n = 0;

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

{

unsigned mask = 1 << 31;

for(int j = 1; j <= 32; j++)

{

cout << (massiv[i] & mask ? "1":"0");

n+=1;

mask >>= 1;

if (n == 1 || n == 11)

{

cout << " ";

}

}

}

cout << '\n';

cout << "Конечный результат: "<< idouble << endl;

cout <<'\n';

}

}

else

{

break;

}

break;

}

case 5:{

flag\_1 = false;

break;

}

default:{

cout << "Неверный номер задания." << endl;

break;

}

}

}

}

case 2:{

const int n = 100;

int arr[n];

int originalArr[n];

srand(static\_cast<unsigned>(time(0)));

int count\_task = 0;

int max\_number = arr[0];

int min\_number = arr[0];

bool flag = true;

bool flag\_sort = false;

system("clear");

while (flag){

int task;

cout << "\n\nВыберите номер задания(1-8) (9 - завершение программы) (10 - идз): ";

cin >> task;

switch(task){

case 1:{

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

arr[i] = -99 + (rand() % 199);

originalArr[i] = arr[i];

}

cout << "\nМассив до сортировки: ";

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

cout << arr[i] << " ";

}

count\_task++;

max\_number = 0;

min\_number = 99;

for (int i = 1; i < n; ++i) {

if (arr[i] > max\_number) max\_number = arr[i];

if (arr[i] < min\_number) min\_number = arr[i];

}

break;

}

case 2:{

if (count\_task == 0){

cout << "\nМассив ещё не был сгенерирован! Попробуйте еще раз!";

break;

}

auto start = steady\_clock::now();

cocktailSort(arr,n);

auto end = steady\_clock::now();

auto result\_sort = duration\_cast<nanoseconds>(end - start);

cout << "\n\nМассив после сортировки: ";

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

cout << arr[i] << " ";

}

cout <<"\n\nВремя сортировки в наносекундах = " <<result\_sort.count();

flag\_sort = true;

break;

}

case 3:{

if (count\_task == 0){

cout << "\nМассив ещё не был сгенерирован! Попробуйте еще раз!";

break;

}

cout << "\n\nВыберите в каком массиве производить поиск(1-сорт/0-несорт)";

int choice;

cin >> choice;

if (choice == 1){

auto start\_search\_sort = steady\_clock::now();

max\_number = arr[99];

min\_number = arr[0];

auto end\_search\_sort = steady\_clock::now();

auto result\_search\_sort = duration\_cast<nanoseconds>(end\_search\_sort - start\_search\_sort);

cout << "\n\nНаибольший элемент массива(сорт): "<< max\_number;

cout << "\nНаименьший элемент массива(сорт): "<< min\_number;

cout << "\n\nВремя поиска макс и мин элементов в отсортированном массиве = " << result\_search\_sort.count() << " наносекунд";

}

else{

/\*

for(int i = 0; i < 100; i++){

cout << originalArr[i]<<" ";

}

\*/

int max\_number = originalArr[0];

int min\_number = originalArr[0];

auto start\_search\_sort = steady\_clock::now();

for (int number : originalArr) {

if (number > max\_number) max\_number = number;

if (number < min\_number) min\_number = number;

}

auto end\_search\_sort = steady\_clock::now();

auto result\_search\_sort = duration\_cast<nanoseconds>(end\_search\_sort - start\_search\_sort);

cout << "\n\nНаибольший элемент массива(несорт): "<< max\_number;

cout << "\nНаименьший элемент массива(несорт): "<< min\_number;

cout << "\n\nВремя поиска макс и мин элементов в отсортированном массиве = " << result\_search\_sort.count() << " наносекунд";

}

break;

}

case 4:{

if (count\_task == 0) {

cout << "nМассив ещё не был сгенерирован! Попробуйте еще раз!";

break;

}

cout << "Выберите в каком массиве хотите произвести поиск(1-сорт/0-несорт): ";

int choice\_4;

cin >> choice\_4;

if (choice\_4 == 1) {

int mid\_num\_sort = (max\_number + min\_number) / 2.0;

cout << "\n\nСреднее значение макс и мин элементов в отсортированном массиве = " << mid\_num\_sort;

vector<int> indices;

auto start\_index\_search\_sort = steady\_clock::now();

int found\_index = binarySearch(arr, mid\_num\_sort, 0, n - 1);

if (found\_index == -1) {

cout << "\nЭлемент не найден в массиве!" << endl;

break;

}

for (int i = found\_index - 1; i >= 0 && arr[i] == mid\_num\_sort; i--) {

indices.push\_back(i);

}

indices.push\_back(found\_index);

for (int i = found\_index + 1; i < n && arr[i] == mid\_num\_sort; i++) {

indices.push\_back(i);

}

auto end\_index\_search\_sort = steady\_clock::now();

auto result\_index\_search\_sort = duration\_cast<nanoseconds>(end\_index\_search\_sort - start\_index\_search\_sort);

cout << "\n\nИндексы элементов среднего значения(сорт) = ";

for (int ind : indices) {

cout << ind << " ";

}

cout << "\nИх кол-во = " << indices.size() << endl;

cout << "\n\nВремя поиска индексов среднего значения в отсортированном массиве = " << result\_index\_search\_sort.count() << " наносекунд";

break;

}

if (choice\_4 == 0){

float mid\_num\_unsort = (max\_number + min\_number) / 2.0;

cout << "\n\nСреднее значение макс и мин элементов в неотсортированном массиве = " << mid\_num\_unsort;

vector<int> indeces;

auto start\_index\_search\_unsort = steady\_clock::now();

int count\_mid = 0;

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

if (mid\_num\_unsort == originalArr[i]){

indeces.push\_back(i);

count\_mid++;

}

}

auto end\_index\_search\_unsort = steady\_clock::now();

auto result\_index\_search\_unsort = duration\_cast<nanoseconds>(end\_index\_search\_unsort - start\_index\_search\_unsort);

cout << "\n\nИндексы элементов среднего значения(несорт) = ";

for (int ind : indeces) {

cout << ind << " ";

}

cout << "\nИх кол-во = "<<count\_mid << endl;

cout << "\n\nВремя поиска индексов среднего значения в неотсортированном массиве = " << result\_index\_search\_unsort.count() << " наносекунд";

break;

}

}

case 5:{

if (count\_task == 0){

cout << "\nМассив ещё не был сгенерирован! Попробуйте еще раз!";

break;

}

int a;

cout << "\nВведите число a ";

cin >> a;

int count\_a = 0;

unordered\_set<int> unique\_numbers;

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

if (arr[i] < a) {

unique\_numbers.insert(arr[i]);

}

}

count\_a = unique\_numbers.size();

cout << "\n\nКол-во чисел в массиве, которые меньше введенного числа a = " << count\_a<< endl;

break;

}

case 6:{

if (count\_task == 0){

cout << "\nМассив ещё не был сгенерирован! Попробуйте еще раз!";

break;

}

int b;

cout << "\nВведите число b ";

cin >> b;

int count\_b = 0;

unordered\_set<int> unique\_numbers;

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

if (arr[i] > b) {

unique\_numbers.insert(arr[i]);

}

}

count\_b = unique\_numbers.size();

cout << "\n\nКол-во чисел в массиве, которые больше введенного числа b = " << count\_b << endl;

break;

}

case 7:{

if (count\_task == 0){

cout << "\nМассив ещё не был сгенерирован! Попробуйте еще раз!";

break;

}

cout << "\n\nВведите число для нахождения его при помощи бинарного поиска: ";

int number;

cin >> number;

auto binary\_searchnumber = steady\_clock::now();

binarySearch(arr, number,0,n-1);

auto binary\_searchnumber\_end = steady\_clock::now();

auto binary\_searchnumber\_time = duration\_cast<nanoseconds>(binary\_searchnumber\_end - binary\_searchnumber);

if (binarySearch(arr, number,0,n-1)){

cout << "\nЭлемент есть в массиве(бинарный)!"<< endl;

}

else{

cout << "\nЭлемент не найден в массиве(бинарный)!"<< endl;

}

cout << "Время выполнения бинарного поиска = " << binary\_searchnumber\_time.count() << " наносекунд";

bool flag = false;

auto searchnumber = steady\_clock::now();

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

if (arr[i] == number){

flag = true;

break;

}

}

auto searchnumber\_end = steady\_clock::now();

auto searchnumber\_time = duration\_cast<nanoseconds>(searchnumber\_end - searchnumber);

if (flag){

cout << "\n\nЭлемент есть в массиве!"<< endl;

}

else{

cout << "\n\nЭлемент не найден в массиве!"<< endl;

}

cout << "Время выполнения переборного поиска = " << searchnumber\_time.count() << " наносекунд";

flag = false;

break;

}

case 8:{

if (count\_task == 0){

cout << "\nМассив ещё не был сгенерирован! Попробуйте еще раз!";

break;

}

cout << "\n\nВведите индексы чисел, которые стоит поменять местами: ";

int index1, index2;

cin >> index1 >> index2;

auto swap\_time = steady\_clock::now();

swap(arr[index1], arr[index2]);

auto swap\_end = steady\_clock::now();

auto swap\_duration = duration\_cast<nanoseconds>(swap\_end - swap\_time);

cout << "\n\nМассив после замены элементов: ";

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

cout << arr[i] << " ";

}

cout << "\n\nВремя замены элементов местами = " << swap\_duration.count() << " наносекунд";

break;

}

case 9:{

flag = false;

break;}

case 10:{

if (count\_task == 0){

cout << "\nМассив ещё не был сгенерирован! Попробуйте еще раз!";

break;

}

cout << "Введите с каким массивом хотите работать: (1 - сорт/0 - несорт): ";

int answer\_idz;

cin >> answer\_idz;

if (answer\_idz == 1){

if (flag\_sort == false){

cout << "Массив еще не был отсортирован, попробуйте снова!";

break;

}

int idz\_number;

cout << "Введите на какое значение будем уменшать каждый нечетный элемент: " << endl;

cin >> idz\_number;

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

if(i % 2 != 0){

arr[i] -= idz\_number;

}

}

cout << "Массив после уменьшения: " << endl;

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

cout << arr[i] << " ";

}

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

if (i % 2 != 0){

arr[i] = arr[i] \* (rand() % 9 + 1);

}

}

cout << '\n';

cout << "Массив после умножение: " << endl;

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

cout << arr[i] << " ";

}

vector <int>counts(10,0); //10 элементов и все равны 0

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

for (int j = 1; j < 10; ++j){

if(arr[i] % j == 0){

counts[j]++;

}

}

}

for(int i = 1; i < 10; i++){

cout << "\nКол-во элементов, которые нацело делятся на "<< i << " равно: "<< counts[i] << endl;

}

}

if (answer\_idz == 0){

int idz\_number;

cout << "Введите на какое значение будем уменшать каждый нечетный элемент: " << endl;

cin >> idz\_number;

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

if(i % 2 != 0){

originalArr[i] -= idz\_number;

}

}

cout << "Массив после уменьшения: " << endl;

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

cout << originalArr[i] << " ";

}

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

if (i % 2 != 0){

originalArr[i] = originalArr[i] \* (rand() % 9 + 1);

}

}

cout << '\n';

cout << "Массив после умножение: " << endl;

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

cout << originalArr[i] << " ";

}

vector <int>counts(10,0);

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

for (int j = 1; j < 10; ++j){

if(originalArr[i] % j == 0){

counts[j]++;

}

}

}

for(int i = 1; i < 10; i++){

cout << endl;

cout << "\nКол-во элементов, которые нацело делятся на "<< i << " равно: "<< counts[i] << endl;

}

}

}

default:{

cout << "Неверный номер задания." << endl;

break;

}

}

}

}

case 3:{

int arr[rows][cols];

float f\_arr[rows][cols];

system ("clear");

bool flag = true;

while (flag) {

cout << endl;

cout << "Введите номер задания (1-4, 5 - завершение программы): ";

int task;

cin >> task;

cout << endl;

switch(task){

case 1:{

cout << "Выберите вывод (1 - спираль, 2 - змейка): ";

int conclusion;

cin >> conclusion;

fillSpiral(&arr[0][0], rows, cols);

if(conclusion == 1){

printSpiral(arr, rows, cols);

} else {

printSnake(arr, rows, cols);

}

break;

}

case 2:{

cout<<"Выберите перестановку(1/2/3/4): ";

int permutation;

cin >> permutation;

if (permutation == 1){

cout << endl << endl;

cout << "Поворот блоков:\n";

usleep(2000000);

change\_A(arr);

printMatrix(arr, rows, cols);

}

if (permutation == 2){

cout << endl << endl;

cout << "Поворот блоков:\n";

usleep(2000000);

change\_B(arr);

printMatrix(arr, rows, cols);

}

if (permutation == 3){

cout << endl << endl;

cout << "Поворот блоков:\n";

usleep(2000000);

change\_C(arr);

printMatrix(arr, rows, cols);

}

if (permutation == 4){

cout << endl;

cout << "Поворот блоков:\n";

usleep(2000000);

change\_D(arr);

printMatrix(arr, rows, cols);

}

break;

}

case 3:{

cout << "Отсортированный массив: "<< endl;

fillSpiralSorted(arr);

printMatrix(arr, rows, cols);

break;

}

case 4:{

cout << "Выберите действие (1 - плюс, 2 - минус, 3 - умножение, 4 - деление): ";

int operation;

cin >> operation;

cout << endl;

if (operation == 1){

cout << "Ввдеите число, на которое увеличится каждое число матрицы: ";

int n;

cin >> n;

plusNumber(&arr[0][0], rows, cols, n);

cout << "Измененная матрица: " << endl;

printMatrix(arr, rows, cols);

}

if (operation == 2){

cout << "Ввдеите число, на которое уменьшится каждое число матрицы: ";

int n;

cin >> n;

minusNumber(&arr[0][0], rows, cols, n);

cout << "Измененная матрица: " << endl;

printMatrix(arr, rows, cols);

}

if (operation == 3){

cout << "Ввдеите число, на которое умножится каждое число матрицы: ";

int n;

cin >> n;

multiplyNumber(&arr[0][0], rows, cols, n);

cout << "Измененная матрица: " << endl;

printMatrix(arr, rows, cols);

}

if (operation == 4){

cout << "Ввдеите число, на которое поделится каждое число матрицы: ";

int n;

cin >> n;

intToFloatMatrix(&arr[0][0], &f\_arr[0][0], rows, cols);

shareNumber(&f\_arr[0][0], rows, cols, n);

}

}

case 5:{

flag = false;

break;

}

default:{

cout << "Неверный номер задания." << endl;

break;

}

}

}

}

case 4:{

int choice;

vector<string> words;

string userInput;

string editedInput;

bool flag = true;

string cleanedInput;

system("clear");

while (flag) {

cout << endl;

cout << "Введите номер задания(1 - 4): ";

cin >> choice;

cout << endl;

cin.ignore();

switch (choice) {

case 1: {

int user\_choice;

cout << "Выберите способ ввода текста(1 - с клавиатуры, 2 - из файла): ";

cin >> user\_choice;

cin.ignore();

if (user\_choice == 1) {

userInput = inputFromKeyboard();

} else if (user\_choice == 2) {

userInput = inputFromFile();

} else {

cout << "Неверный выбор. Пожалуйста, попробуйте снова." << endl;

return 0;

}

if (isValidInput(userInput, words)) {

cout << "Введенные слова: ";

for (const auto& word : words) {

cout << word << " ";

}

cout << endl;

} else {

cout << "Нет корректных слов для вывода." << endl;

}

break;

}

case 2: {

editedInput = removeExtraSpaces(userInput);

editedInput = removeExtraPunctuation(editedInput);

editedInput = toLowerCase(editedInput);

cout << "Отредактированный текст: " << editedInput << endl;

break;

}

case 3: {

string charsToRemove;

cout << "Введите набор символов для удаления: ";

getline(cin, charsToRemove);

cout << "Введенные слова (без цифр и с удаленными символами): ";

vector<string> editedWords;

istringstream iss(editedInput);

string word;

while (iss >> word) {

editedWords.push\_back(word);

}

for (const auto& word : editedWords) {

if (!containsDigits(word)) {

string cleanedWord = removeCharacters(word, charsToRemove);

cleanedInput += cleanedWord + " ";

cout << cleanedWord << " ";

}

}

cout << endl;

break;

}

case 4:{

string substring;

cout << "Введите подстроку для поиска: ";

getline(cin, substring);

vector<int> bmPositions = boyerMooreSearch(cleanedInput, substring);

cout << "Результаты поиска Бойера-Мура: ";

for (int pos : bmPositions) {

cout << pos << " ";

}

cout << endl;

break;

}

default:{

cout << "Неверный номер задания." << endl;

break;

}

}

}

}

default:{

cout << "Неверный номер задания." << endl;

break;

}

}

}

}