**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++"

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. | Тимошенков И.М. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2022

**Цель работы.**

Создать интерфейс работы с несколькими программами в одном приложении.

**Основные теоретические положения.**

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.

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.

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.

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.

Общая суть сортировки вставками такова:

1)    Перебираются элементы в неотсортированной части массива.

2)    Каждый элемент вставляется в отсортированную часть массива на то место, где он должен находится.

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.

В изучаемых нами языках программирования между массивами и указателями имеется очень тесная связь.

Когда мы определяем в программе некоторый массив, например, переменная **Arr** без индексов представляет собой указатель на первый элемент массива в данном случае из 10 целых чисел (содержит адрес первого элемента массива). Если вывести на экран значение переменной **Arr**, мы увидим некоторое целое значение в шестнадцатеричном формате, соответствующее адресу первого элемента этого массива.

**Замечание**. Именно по этой причине в языке C++ отсутствует операция присвоения сразу всех значений одного массива другому (в некоторых других языках, например, в Pascal такая возможность имеется). Действительно, если имеются два массива

то попытка выполнить присвоение **A1 = A2**привела бы к тому, что переменная **A1** стала бы указывать на ту же область памяти, что и переменная **A2** (мы скопировали адрес из **A2**в **A1**, а не содержимое одного массива в другой).  Адрес, который хранился ранее в переменной **A1,** был бы утерян, что привело бы к утечке памяти (для десяти элементов массива **A1** в памяти было выделено место, но теперь мы “забыли”, где оно находится, то есть потеряли память). По этой причине подобные операции с массивами в языке C++ запрещены. Более того, запрещены любые изменения значения переменной массива.

Указателю, имеющему такой же базовый тип, как и элементы массива, можно присвоить массив следующим образом. Но обратное присвоение выполнить невозможно.

Такое присвоение невозможно, поскольку переменная массива – это константа, изменение которой запрещено.

Так как переменная массива является указателем на первый элемент массива, появляются дополнительные возможности по работе с массивами на основе использования арифметики указателей. Например, чтобы получить 5–й элемент массива **Arr**можно воспользоваться одним из следующих выражений:

**Arr[4]**или   \*(**Arr + 4)**или**\*( p + 4)**

Первое выражение – это пример обычной индексации элементов массива. Во втором и третьем выражениях мы использовали арифметику указателей и с помощью операции + получили адрес пятого элемента массива. Затем с помощью операции \* взяли значение по этому адресу и получили значение 5-го элемента массива. Обратите внимание на скобки в этих выражениях, если их не поставить и написать \***Arr + 4**или **\*p + 4**, то эти выражения будут равны значению первого элемента массива увеличенного на 4, так как операция \* имеет больший приоритет, чем операция +.

Вот пример фрагмента программы для работы с массивом с помощью обычной индексации элементов массива. Этот фрагмент обеспечивает ввод элементов целочисленного массива с клавиатуры, вычисление квадратов значений элементов массива, а затем вывод элементов массива на экран:

Использование арифметики указателей при работе с массивами приводит обычно к уменьшению объема генерируемого кода программы и к уменьшению времени ее выполнения, то есть к увеличению быстродействия.

Поскольку указатель и имя массива, в большой степени, взаимозаменяемы, указатели можно индексировать, как обычные массивы. Можно создавать и массивы указателей.

Массив **M** – это трехэлементный массив указателей на целые значения, то есть каждый элемент этого массива представляет собой указатель на целое.

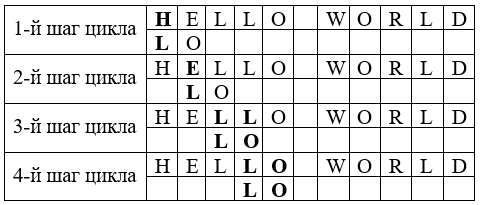
С помощью массивов указателей можно моделировать различные интересные конструкции данных. Например, пусть имеется квадратная матрица размерности 5 х 5 симметричная относительно главной диагонали. Для ее однозначного представления достаточно хранить в памяти  не все 25 элементов этой матрицы, а только 15 (например, элементы под главной диагональю вместе с элементами главной диагонали). Для этого можно предложить следующую конструкцию:

А это пятиэлементный массив указателей на символы, инициализированный некоторыми текстовыми строками:

Как это работает: когда компилятор встречает в программе некоторый текст, заключенный в кавычки, в памяти создается символьный массив соответствующей этому тексту длины и адрес этого символьного массива присваивается соответствующему элементу – указателю массива **Words**.

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.



На первых двух итерациях цикла сравниваемые буквы не будут совпадать. На третьей же итерации, совпал символ ‘L’, это означает, что теперь нужно сравнивать следующий символ подстроки со следующим символом строки. Видно, что символы отличаются, поэтому алгоритм продолжает свою работу. На четвертой же итерации подстрока была найдена.

Если представить, что исходная строка непорядок больше и подстрока находится в конце строки (или вовсе отсутствует), то сразу видны минусы данного алгоритма.

Одним из самых популярных алгоритмов, который работает быстрее, чем приведенный выше алгоритм, является алгоритм Кнута-Морриса-Пратта (КМП). Идея заключается в том, что не нужно проходить и сравнивать абсолютно все символы строки, если известны символы, которые есть и в строке, и в подстроке.

Суть алгоритма: дана подстрока S и строка T. Требуется определить индекс, начиная с которого образец S содержится в строке T. Если S не содержится в T, необходимо вернуть индекс, который не может быть интерпретирован как позиция в строке.



Хоть алгоритм и работает быстрее, по-прежнему необходимо сначала пройти всю строку, чтобы определить префиксы или суффиксы (вхождение (индексы) символов).

Алгоритм Бойера-Мура в отличие от КМП полностью не зависим и не требует заранее проходить по строке. Этот алгоритм считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке.

Преимущество этого алгоритма в том, что ценной некоторого количества предварительных вычислений над подстрокой (но не над исходной строкой, в которой ведётся поиск), подстрока сравнивается с исходным текстом не во всех позициях (пропускаются позиции, которые точно не дадут положительный результат).

Поиск подстроки ускоряется благодаря созданию таблиц сдвигов. Сравнение подстроки со строки начинается с последнего символа подстроки, а затем происходит прыжок, длина которого определяется по таблице сдвигов. Таблица сдвигов строится по подстроке так чтобы перепрыгнуть максимальное количество символов строки и не пропустить вхождение подстроки в строку.

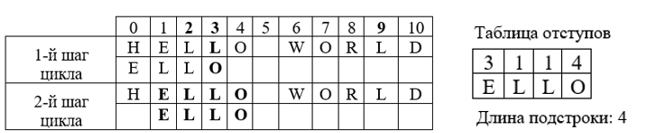
Правила построения таблицы сдвигов:

1)    Значение элемента таблицы равно удаленности соответствующего символа от конца шаблона (подстроки).

2)    Если символ встречается более одного раза, то применятся значение, соответствующее символу, наиболее близкому к концу шаблона.

3)    Если символ в конце шаблона встречается 1 раз, ему соответствует значение, равное длине образа; если более одного раза – значение, соответствующее символу, наиболее близкому к концу образа.

4)    Для символов, отсутствующих в образе, применяется значение, равное длине шаблона.



Сначала была построена таблица отступов и подсчитана длина подстроки. Затем начинается алгоритм поиска подстроки в строке. Сравнивает символ ‘L’ строки и ‘O’ подстроки. Элементы не совпадают, поэтому необходимо определить длину отступа. Символ ‘L’ присутствует в таблице отступа, длина отступа равняется 1. Подстрока смещается на 1 символ вперед. На следующей итерации подстрока найдена.

**Постановка задачи.**

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

**Выполнение работы.**

Код программы.

#include<ctime> // подключение ctime для работы с srand

#include<cmath> // подключение cmath для работы с логарифмами

#include<chrono> // подключение chrono

#include <iostream>

#include <Windows.h>

#include <iomanip>

#include <string>

#include <fstream>

#include <conio.h>

using namespace std;

using namespace chrono; // подключение пространства имен chrono

//-----------------------------Практика:1-------------------------------------

void integer() {

int mask;

mask = 1 << (sizeof(int) \* 8) - 1; // создаем маску 1000000... и переводим байты в биты

// сдвигаем 1 на последний бит

int number; // инициализайия для введенного числа

cout << "Введите целое число: ";

cin >> number;

for (int i = 0; i < sizeof(int) \* 8; i++) // условие цикла

{

if (i == 1) // разделение разрядов на знак под символ и под значение

cout << ' ';

if (number & mask) // сравнение с маской, если значение после конъюкции не 0, то 1 на экран

cout << '1';

else

cout << '0';

number <<= 1; // сдвиг числа влево

}

}

void floatNum() {

int mask;

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

cout << "\nВведите вещественное число: ";

union {

float floatValue; // запись двух переменных в один адрес

int numberForFloat;

};

cin >> floatValue;

for (int i = 0; i < sizeof(int) \* 8; i++) // условие цикла

{

if (i == 1 || i == 9) // выделение мантисы и порядка 0 000000000 1000011100111010000000

cout << ' ';

if (numberForFloat & mask) // сравнение с маской, если значение после конъюкции не 0, то 1 на экран

cout << '1';

else

cout << '0';

numberForFloat <<= 1; // сдвиг числа влево

}

}

void doubleNum() {

int mask;

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

cout << "\nВведите вещественное число: ";

union {

double doubleValue; // запись двух переменных в один адрес

int listForDouble[2]; // выделения массива, отводится 8 байт на 2 целочисленных числа

};

cin >> doubleValue; // после ввода, память выделенная для массива заполняется

// так как массив заполняет ячейки по порядку, то сначала выведем алгоритм для 0 элемента

for (int i = 0; i < sizeof(int) \* 8; i++) // условие цикла

{

if (i == 1 || i == 12) // разделение разрядов на знак под символ и под значение

cout << ' ';

if (listForDouble[1] & mask) // сравнение с маской, если значение после конъюкции не 0, то 1 на экран

cout << '1';

else

cout << '0';

listForDouble[1] <<= 1; // сдвиг числа влево

}

for (int i = 0; i < sizeof(int) \* 8; i++) // условие цикла

{

if (listForDouble[0] & mask) // сравнение с маской, если значение после конъюкции не 0, то 1 на экран

cout << '1';

else

cout << '0';

listForDouble[0] <<= 1; // сдвиг числа влево

}

}

void sizes() {

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

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

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

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

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

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

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

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

}

void printMenu1() {

system("CLS");

cout << "-----------|Представление типов данных в памяти|-----------\n";

cout << "1-Вывести размер типов данных\n";

cout << "2-Вывести представление в памяти числа типа int\n";

cout << "3-Вывести представление float\n";

cout << "4-Вывести представление double\n";

}

int menu1() {

bool exit = false;

while (!exit) {

printMenu1();

char ch;

ch = \_getch();

switch (ch) {

case '1':

sizes();

cout << "Backspace для возвращения";

ch = \_getch();

break;

case '2':

integer();

cout << '\n' << "Backspace для возвращения";

ch = \_getch();

break;

case '3':

floatNum();

cout << '\n' << "Backspace для возвращения";

ch = \_getch();

break;

case '4':

doubleNum();

cout << '\n' << "Backspace для возвращения";

ch = \_getch();

break;

case 8:

return 0;

break;

default:

break;

}

}

}

//-----------------------------Практика:2-------------------------------------

void binarySearch(int\* A, int size, int number) //алгоритм бинарного поиска

{

int step\_size = size / 2; // часть размера на которую сдвигается область поиска числа

int bin\_size = size / 2; // середина массива

int flagNumber = 0; // флаг, который отслеживает найденное число

float log = log2(size); // количество шагов, необходимое для поиска числа

for (int i = 0; i < int(ceil(log)); i++)

{

if (step\_size != 1) // так как шаг постепенно стримиться к нулю, то тут реализована проверка

{

step\_size = step\_size / 2; // если шаг больше 1, то он делиться, если нет - то он так и остается равен 1

}

if (number > A[bin\_size]) // если число больше числа, на которое в данный момент указывает bin\_size,

{

bin\_size = bin\_size + step\_size; // то прибавляем некоторую часть размера

}

else if (number < A[bin\_size])

{

bin\_size = bin\_size - step\_size; // иначе - отнимаем

}

if (number == A[bin\_size]) // число найдено и алгоритм прекращает работу

{

flagNumber = A[bin\_size];

break;

}

}

if (flagNumber != 0)

{

cout << "Ваше число есть в массиве \n";

}

else

{

cout << "Вашего числа нет в массиве \n";

}

}

void changeElements(int\* A) // работа по смене чисел через функцию swap

{

int indx1;

int indx2;

cout << "Введите индексы элементов, которые хотели бы поменять местами \n";

cout << "Индекс\_1 - ";

cin >> indx1;

cout << '\n' << "Индекс\_2 - ";

cin >> indx2;

steady\_clock::time\_point s = steady\_clock::now();

cout << '\n' << "Дело сделано \n";

swap(A[indx1], A[indx2]);

steady\_clock::time\_point e = steady\_clock::now();

int t = duration\_cast<nanoseconds>(e - s).count();

cout << t << " наносекунд \n";

}

int stupidSearch(int\* A, int size) // обычный линейный поиск

{

cout << "Загружается алгоритм линейного поиска... \n";

cout << "Введите число для поиска при помощи линейного \n";

int number;

cin >> number;

steady\_clock::time\_point st = steady\_clock::now();

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

{

if (A[i] == number)

{

cout << "Ваше число в массиве \n";

steady\_clock::time\_point en = steady\_clock::now();

cout << duration\_cast<nanoseconds>(en - st).count() << " наносекунд \n";

return 0;

}

}

cout << "Вашего числа нет в массиве \n";

steady\_clock::time\_point end = steady\_clock::now();

cout << duration\_cast<nanoseconds>(end - st).count() << " наносекунд \n";

return 0;

}

void quicksort(int\* A, int end, int begin) // алгоритм быстрой сортировки

{

int mid;

int f = begin;

int l = end;

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

while (f < l)

{

while (A[f] < mid)

{

f++;

}

while (A[l] > mid)

{

l--;

}

if (f <= l)

{

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

f++;

l--;

}

}

if (begin < l) quicksort(A, l, begin);

if (f < end) quicksort(A, end, f);

}

void maxMinSorted(int\* A, int size) // поиск в отсортированном массиве

{

cout << "Отсортированный массив \n";

steady\_clock::time\_point s = steady\_clock::now();

cout << A[0] << ' ' << A[size - 1] << ' '; // программа обращается к последнему и первому элементу

steady\_clock::time\_point e = steady\_clock::now();

cout << duration\_cast<nanoseconds>(e - s).count() << " наносекунд \n";

}

void maxMinNoSorted(int\* A, int size) // поиск в неотсортированном массиве

{

cout << "Неотсортированный массив \n";

int max = 0;

int min = 10000000;

steady\_clock::time\_point s = steady\_clock::now();

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

{

if (A[i] > max)

{

max = A[i];

}

if (A[i] < min)

{

min = A[i];

}

}

cout << min << ' ' << max << ' ';

steady\_clock::time\_point e = steady\_clock::now();

cout << duration\_cast<nanoseconds>(e - s).count() << " наносекунд \n";

}

void averageVal(int\* A, int size) // кол-во элементов и их индексы, которые равны среднему max и min

{

int count = 0;

int average = (A[0] + A[size - 1]) / 2;

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

{

if (A[i] == average)

{

count++;

cout << "Ин." << i << ' ';

}

}

cout << "Кол - во: " << count << '\n';

}

void lessA(int\* A, int size) // кол-во чисел меньших a

{

int a;

cout << "Введите число: ";

cin >> a;

int count = 0;

cout << '\n';

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

{

if (A[i] < a)

{

count++;

}

}

cout << count << '\n';

}

void bigger(int\* A, int size) // кол-во чисел больших b

{

int a;

cout << "Введите число: ";

cin >> a;

int count = 0;

cout << '\n';

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

{

if (A[i] > a)

{

count++;

}

}

cout << count << '\n';

}

//-----------------------------Практика:3-------------------------------------

void zigzag(int arr3[10][10], int arr[10][10], int size, int spacey) { //слева пустой массив

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

destCoord.X = 0;

destCoord.Y = 0;

SetConsoleCursorPosition(hStdout, destCoord);

int columnChanges = 0;

int stepZigzag = 0;

int\* ppp = &arr[0][0];

int spacex = 0;

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

if (stepZigzag % 2 == 0) {

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

arr3[g][columnChanges] = \*(ppp);

destCoord.X = columnChanges + spacex;

destCoord.Y = g + g + 1 + spacey;

SetConsoleCursorPosition(hStdout, destCoord); // отвечает за вывод на экран чисел в нужном порядке

Sleep(70);

cout << \*ppp;

ppp += 1;

}

spacex += 3;

}

else {

for (int jj = size - 1; jj > -1; jj--) {

arr3[jj][columnChanges] = \*(ppp);

destCoord.X = columnChanges + spacex;

destCoord.Y = jj + jj + 1 + spacey;

SetConsoleCursorPosition(hStdout, destCoord);

Sleep(70);

cout << \*ppp;

ppp += 1;

}

spacex += 3;

}

columnChanges += 1;

stepZigzag += 1;

}

destCoord.X = 0;

destCoord.Y = spacey + 18;

SetConsoleCursorPosition(hStdout, destCoord);

}

// изменение положения блоков матрицы, которое производится через индексы, меняет левые и правые блоки

void left\_rightSwap(int arr[10][10], int size) {

for (int h = 0; h < size / 2; h++) {

for (int k = 0; k < size / 2; k++) {

swap(arr[k][h], arr[k][h + (size / 2)]);

swap(arr[k + (size / 2)][h], arr[k + (size / 2)][h + (size / 2)]);

}

}

}

// меняет верхние и нижние блоки

void up\_downSwap(int arr[10][10], int size) {

for (int h = 0; h < size / 2; h++) {

for (int k = 0; k < size / 2; k++) {

swap(arr[h][k], arr[h + (size / 2)][k]);

swap(arr[h][k + (size / 2)], arr[h + (size / 2)][k + (size / 2)]);

}

}

}

// меняет блоки крест-накрест

void left\_down\_rightSwap(int arr2[10][10], int size) {

for (int h = 0; h < size / 2; h++) {

for (int k = 0; k < size / 2; k++) {

swap(arr2[k][h], arr2[k + (size / 2)][h + (size / 2)]);

swap(arr2[k + (size / 2)][h], arr2[k][h + (size / 2)]);

}

}

}

// меняет блоки по-спирали (пункт а)

void swapBySpiral(int arr3[10][10], int nn) {

for (int h = 0; h < nn / 2; h++) {

for (int k = 0; k < nn / 2; k++) {

swap(arr3[h][k + (nn / 2)], arr3[h + (nn / 2)][k + (nn / 2)]);

}

}

for (int h = 0; h < nn / 2; h++) {

for (int k = 0; k < nn / 2; k++) {

swap(arr3[k][h], arr3[k][h + (nn / 2)]);

}

}

for (int h = 0; h < nn / 2; h++) {

for (int k = 0; k < nn / 2; k++) {

swap(arr3[h][k], arr3[h + (nn / 2)][k]);

}

}

}

//выводит матрицу на экран

void matrixOutPut(int arr[10][10], int size) {

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

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

cout << arr[i][j] << '\t';

}

cout << '\n';

}

cout << '\n';

cout << '\n';

cout << '\n';

}

// сортировка 2мерного массива

void bubbleSort(int arr[10][10], int size)

{

if (size == 6) {

int arr2[6][6];

int\* s = &arr2[0][0];

int\* end = &arr[9][9];

for (int\* i = &arr[0][0]; i < end; (++i, s++)) {

if (\*i < 0) {

s--;

}

else {

\*s = \*i;

}

}

int\* end2 = &arr2[size - 1][size - 1];

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

for (int\* j = &arr2[0][0]; j < end2; j++) {

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

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

}

}

s = &arr2[0][0];

int k = 1;

for (int\* i = &arr[0][0]; i < end + 1; (i++, s++)) {

if (\*i < 0) {

s--;

}

else {

\*i = \*s;

}

}

}

if (size == 8) {

int arr2[8][8];

int\* s = &arr2[0][0];

int\* end = &arr[9][9];

for (int\* i = &arr[0][0]; i < end; (++i, s++)) {

if (\*i < 0) {

s--;

}

else {

\*s = \*i;

}

}

int\* end2 = &arr2[size - 1][size - 1];

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

for (int\* j = &arr2[0][0]; j < end2; j++) {

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

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

}

}

s = &arr2[0][0];

int k = 1;

for (int\* i = &arr[0][0]; i < end + 1; (i++, s++)) {

if (\*i < 0) {

s--;

}

else {

\*i = \*s;

}

}

}

if (size == 10) {

int\* end = &arr[size - 1][size - 1];

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

for (int\* j = &arr[0][0]; j < end; j++) {

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

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

}

}

}

}

// создает матрицу

void creatMatrix(int arr2[10][10], int size) {

int\* pointerOnMat = &arr2[0][0];

int\* end = &arr2[size - 1][size - 1];

for (int\* pointerOnMat = &arr2[0][0]; pointerOnMat <= end; pointerOnMat++)

{

\*pointerOnMat = 0 + rand() % (size \* size) + 1;

}

}

// умножает матрицу на заданное число

void multiplyMatrix(int arr[10][10], int size) {

cout << "Введите число на которое хотите умножить матрицу: ";

int number;

cin >> number;

int\* pointerOnMat = &arr[0][0];

for (int\* end = pointerOnMat + (size \* size); pointerOnMat != end; pointerOnMat++)

{

\*pointerOnMat \*= number;

}

}

// прибавляет к матрице число

void addMatrix(int arr[10][10], int size) {

cout << "Введите число которое хотите прибавить к матрице: ";

int number;

cin >> number;

int\* pointerOnMat = &arr[0][0];

for (int\* end = pointerOnMat + (size \* size); pointerOnMat != end; pointerOnMat++)

{

\*pointerOnMat += number;

}

cout << '\n';

cout << '\n';

cout << '\n';

}

// делит матрицу на число

void divideMatrix(int arr[10][10], int size) {

cout << "Введите число на которое хотите разделить матрицу" << '\n';

int number;

cin >> number;

int\* pointerOnMat = &arr[0][0];

for (int\* end = pointerOnMat + (size \* size); pointerOnMat != end; pointerOnMat++)

{

\*pointerOnMat /= number;

}

}

// вычитает из матрицы число

void subtractMatrix(int arr[10][10], int size) {

cout << "Введите число на которое хотите уменьшить матрицу" << '\n';

int number;

cin >> number;

int\* pointerOnMat = &arr[0][0];

for (int\* end = pointerOnMat + (size \* size); pointerOnMat != end; pointerOnMat++)

{

\*pointerOnMat -= number;

}

}

//-----------------------------Практика:4-------------------------------------

//возвращает настоящую длину массива, нужно для оптимизации массива

int realLength(string\* arr, int size) {

int length = 0;

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

if (arr[i] != "NULL") {

length++;

}

else {

return length;

}

}

}

//разделяет строку в массив по знаку

string\* split(string str, char delim, int& size) {

size = 1000;

str += delim;

string\* array = new string[size];

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

array[i] = "NULL";

}

string word = "";

int k = 0;

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

if (str[i] != delim) {

word += str[i];

}

else {

array[k] = word;

k++;

word = "";

}

}

size = realLength(array, size);

string\* newArr = new string[size];

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

newArr[i] = array[i];

}

delete[] array;

return newArr;

}

//объединяет массив строк в строку чреез знак

string join(string\* str, char del, int& size) {

string stroka = "";

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

stroka += str[i] + del;

}

size = stroka.length();

return stroka;

}

//удаляет вообще все знаки, нужно для заданий, где выводятся только слова

void deleteAllSignesAndDots(string& str) {

while (str.find(",") != -1) {

str.replace(str.find(","), 1, " ");

}

while (str.find("?") != -1) {

str.replace(str.find("?"), 1, " ");

}

while (str.find("!") != -1) {

str.replace(str.find("!"), 1, " ");

}

while (str.find("-") != -1) {

str.replace(str.find("-"), 1, " ");

}

while (str.find(".") != -1) {

str.replace(str.find("."), 1, " ");

}

while (str.find(")") != -1) {

str.replace(str.find(")"), 1, " ");

}

while (str.find("(") != -1) {

str.replace(str.find("("), 1, " ");

}

while (str.find(" ") != -1) {

str.replace(str.find(" "), 2, " ");

}

}

//удаляет лишние знаки

void deleteSignes(string& str, int& size) {

while (str.find(" ") != -1) {

str.replace(str.find(" "), 2, " ");

}

while (str.find(",,") != -1) {

str.replace(str.find(",,"), 2, ",");

}

while (str.find("??") != -1) {

str.replace(str.find("??"), 2, "?");

}

while (str.find("!!") != -1) {

str.replace(str.find("!!"), 2, "!");

}

while (str.find("--") != -1) {

str.replace(str.find("--"), 2, "-");

}

while (str.find("((") != -1) {

str.replace(str.find("(("), 2, "(");

}

while (str.find("))") != -1) {

str.replace(str.find("))"), 2, ")");

}

size = str.length();

}

//приводит слово к нижнему регистру

void redactorOfRegister(string& str) {

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

if (str[i] <= 'Z' && str[i] >= 'A')

str[i] -= ('Z' - 'z');

else if (str[i] <= 'Я' && str[i] >= 'А')

str[i] -= ('Я' - 'я');

}

}

//делает из символа заглавную букву

void mainLetter(char& a) {

if (a <= 'z' && a >= 'a')

a -= ('z' - 'Z');

else if (a <= 'я' && a >= 'а')

a -= ('я' - 'Я');

}

//удаляет все лишние точки, кроме знака моноготочия

void deleteDots(string& str, int& size) {

while (str.find("...") != -1) {

if (str[str.find("...") + 3] != '.' and str[str.find("...") - 1] != '.' and str[str.find("...") - 1] != '<' and str[str.find("...") + 3] != '<') {

str.replace(str.find("..."), 3, "\*");

}

else {

str.replace(str.find("..."), 3, "<");

}

}

while (str.find("<") != -1) {

str.replace(str.find("<"), 1, "...");

}

while (str.find("..") != -1) {

str.replace(str.find(".."), 2, ".");

}

while (str.find("\*") != -1) {

str.replace(str.find("\*"), 1, "...");

}

size = str.length();

}

//редактирует строку в нормальный вид

string redactorOfSentances(string& str, int& size) {

deleteSignes(str, size);

deleteDots(str, size);

string\* b = split(str, ' ', size);

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

redactorOfRegister(b[i]);

}

string newStr = join(b, ' ', size);

b = split(newStr, '.', size);

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

if (b[i][0] != ' ') {

mainLetter(b[i][0]);

}

else {

mainLetter(b[i][1]);

}

}

newStr = join(b, '.', size);

newStr.replace(newStr.length() - 1, 1, "");

return newStr;

delete[] b;

}

//проверяет является ли символ числом

bool isDigit(string str) {

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

if (str[i] <= '9' && str[i] >= '0') {

return false;

}

}

}

//функция для первого задания

void problem1(string str, int size) {

deleteAllSignesAndDots(str);

string\* b = split(str, ' ', size);

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

if (isDigit(b[i])) {

redactorOfRegister(b[i]);

cout << b[i] << " ";

}

}

} //функция для второго задания

void problem2(string str, int size) {

deleteAllSignesAndDots(str);

string\* b = split(str, ' ', size);

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

redactorOfRegister(b[i]);

cout << b[i] << '\n';

}

}

int getNumberOfSize(int\* arr, string str, char a, int size) { // возвращает число сдвига согласно букве из таблицы сдвигов

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

if (str[size - arr[i] - 1] == a) {

return arr[i];

}

}

return arr[size - 1]; // возвращает номер сдвига для любого элемента или для последнего, если он не встречается нигде кроме как в конце

}

// функция, которая проверяет есть ли проверяемый элемент в массиве уже проверенных элементов

int isElement(string str, int\* a, int j, int size) {

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

if (a[i] < str.length() and a[i] >= 0) {

if (str[a[i]] == str[j]) {

return a[i];

}

};

}

return -1;

}

void findInText(string str, string text) { //поиск по алгоритму Бойера-Мура

int size = str.length();

int\* arrayOfIndexes = new int[size]; //массив, который сохраняет индексы уже обработанных символов в таблице

int\* arrayOfSizes = new int[size]; //сама таблица

// цикл формирования таблицы

for (int i = size - 2; i >= 0; i--) {

int prevIndex = isElement(str, arrayOfIndexes, i, size);

if (prevIndex != -1) {

arrayOfSizes[i] = size - (prevIndex + 1);

}

else {

arrayOfSizes[i] = size - (i + 1);

arrayOfIndexes[i] = i;

}

}

//рассчитывание отступа для последнего элемента

int prevIndexOfLast = isElement(str, arrayOfIndexes, size - 1, size);

if (prevIndexOfLast != -1) {

arrayOfSizes[size - 1] = size - (prevIndexOfLast + 1);

}

else {

arrayOfSizes[size - 1] = size;

}

int\* indexes = new int[100];

int point = -(int)str.length() + 1;

int counterForIndexes = 0;

bool isFinded = false;

while ((point + str.length()) <= text.length()) {

int k = str.length() - 1;

int numberOfMatches = 0;

for (int pointerRev = point + str.length() - 1; point <= pointerRev; pointerRev--) {

if (text[pointerRev] == str[k]) {

numberOfMatches += 1;

k--;

if (numberOfMatches == str.length()) {

indexes[counterForIndexes] = point;

counterForIndexes += 1;

point += str.length();

}

}

else {

if (str.find(text[pointerRev]) != -1) {

point += getNumberOfSize(arrayOfSizes, str, text[pointerRev], str.length());

break;

}

else {

point += str.length();

break;

}

}

}

} for (int i = 0; i < counterForIndexes; i++) {

for (int j = 0; j < str.length(); j++) {

mainLetter(text[indexes[i] + j]);

}

}

delete[] indexes;

cout << text;

}

void lineSearch(string str, string text) { // линейный поиск

int\* indexes = new int[100];

int e = 0;

for (int i = 0; i < text.length() - str.length() + 1; i++) {

int k = 0;

for (int j = 0; j < str.length(); j++) {

if (str[j] == text[i + j]) {

k += 1;

}

}

if (k == str.length()) {

indexes[e++] = i;

}

}

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

for (int j = 0; j < str.length(); j++) {

mainLetter(text[indexes[i] + j]);

}

}

cout << text;

delete[] indexes;

}

int menu4() {

bool exit = false;

while (!exit) {

system("CLS");

string text;

string flag;

cout << "Как хотите прочесть строку (из файла - введите 1)/(из консоли - введите 2)\n";

while (flag != "2" and flag != "1") {

getline(cin, flag);

if (flag != "2" and flag != "1") {

cout << "Вы ввели неправильную цифру. Повторите ввод\n";

}

if (flag[0] == 8) {

return 0;

}

}

if (flag == "1") {

system("CLS");

ifstream fin;

string path;

cout << "Введите имя файла и расширение, файл должен быть формата txt, сохраненный в кодировке ANSI: ";

getline(cin, path);

fin.open(path);

if (!fin.is\_open()) {

cout << "Ошибка открытия" << '\n';

}

else {

text = "";

string line;

while (getline(fin, line)) {

text += line + " ";

}

}

fin.close();

}

else if (flag == "2") {

system("CLS");

cout << "Введите строку: ";

getline(cin, text);

}

cout << "------------------------------------|Задание-1|------------------------------------";

cout << '\n';

problem1(text, text.length());

cout << '\n';

cout << "------------------------------------|Задание-2|------------------------------------";

cout << '\n';

problem2(text, text.length());

cout << '\n';

int size = text.length();

text = redactorOfSentances(text, size);

cout << "Отредактированное предложение: " << text << '\n';

cout << "Введите строку которую хотите найти: ";

string str;

getline(cin, str); cout << '\n';

cout << "---Работа линейного поиска---";

cout << '\n';

lineSearch(str, text); cout << '\n' << '\n';

cout << "---Работа алгоритма Бойера-Мура---";

cout << '\n';

findInText(str, text);

cout << '\n';

cout << "Нажмите backspace, чтобы выйти из программы, чтобы повторить программу, нажмите любую клавишу.";

int ch;

ch = \_getch();

if (ch == 8) {

exit = true;

}

}

} int menu3() {

bool exit = false;

while (!exit) {

int spacey = 0;

system("CLS");

srand(time(0));

int size;

char sz = '0';

cout << "Нажмите цифру 6 8 или 10, чтобы выбрать размер матрицы: ";

cout << '\n';

while (sz != '8' and sz != '6' and sz != '10') {

sz = \_getch();

if (sz != '8' and sz != '6' and sz != '10') {

spacey += 2;

cout << "Вы ввели неправильный размер матрицы \n";

}

if (sz == 8) {

return 0;

}

}

switch (sz)

{

case '8':

size = 8;

break;

case '6':

size = 6;

break;

case '10':

size = 10;

break;

}

int arr[10][10];

int arr2[10][10];

creatMatrix(arr, size);

zigzag(arr2, arr, size, spacey);

cout << "Выберете, как хотите поменять блоки матрицы: \n";

cout << "1 - По спирали\n";

cout << "2 - Верхине и нижние\n";

cout << "3 - Левые и правые\n";

cout << "4 - Крест - накрест\n";

cout << '\n';

char s = '0';

while (s != '1' and s != '2' and s != '3' and s != '4') {

s = \_getch();

if (s != '1' and s != '2' and s != '3' and s != '4') {

cout << "Вы ввели неправильное значение \n";

}

if (s == 8) {

return 0;

}

}

switch (s)

{

case '1':

swapBySpiral(arr2, size);

matrixOutPut(arr2, size);

break;

case '2':

up\_downSwap(arr2, size);

matrixOutPut(arr2, size);

break;

case '3':

left\_rightSwap(arr2, size);

matrixOutPut(arr2, size);

break;

case '4':

left\_down\_rightSwap(arr2, size);

matrixOutPut(arr2, size);

break;

}

cout << "Сортировка матрицы \n";

bubbleSort(arr2, size);

matrixOutPut(arr2, size);

cout << "Выберете операцию над матрицей: \n";

cout << "1 - Прибавление числа\n";

cout << "2 - Умножение на другое число\n";

cout << "3 - Вычитание\n";

cout << "4 - Деление\n";

char ss = '0';

while (ss != '1' and ss != '2' and ss != '3' and ss != '4') {

ss = \_getch();

if (ss != '1' and ss != '2' and ss != '3' and ss != '4') {

cout << "Вы ввели неправильное значение \n";

}

if (ss == 8) {

return 0;

}

}

switch (ss)

{

case '1':

addMatrix(arr2, size);

matrixOutPut(arr2, size);

break;

case '2':

multiplyMatrix(arr2, size);

matrixOutPut(arr2, size);

break;

case '3':

subtractMatrix(arr2, size);

matrixOutPut(arr2, size);

break;

case '4':

divideMatrix(arr2, size);

matrixOutPut(arr2, size);

break;

}

cout << "Нажмите backspace, чтобы выйти из программы, чтобы повторить программу, нажмите любую клавишу.";

int ch;

ch = \_getch();

if (ch == 8) {

exit = true;

}

}

} void printMenu() {

system("CLS");

cout << " ----------|Выберете пункт|---------- \n";

cout << " Представление числа в памяти компьютера. \n";

cout << " Работа с массивами, сортровка массивов. \n";

cout << " Двумерные массивы и указатели. \n";

cout << " Поиск подстроки в строке, работа со строками. \n";

cout << " Выйти \n";

}

void menu2() {

while (1)

{

system("CLS");

srand(time(0));

cout << "------Выберите необходимую операцию------ \n";

const int size = 100;

char input;

int A[size] = {};

int flagarray = 0;

int flagsorted = 0;

int Ansort[size] = {};

int flag = 0;

cout << "Нажмите 1, чтобы создать массив длины 100 \n";

cout << "Нажмите 2, чтобы отсортировать массив длины 100 \n";

cout << "Нажмите 3, чтобы найти максимальный и минимальный элемент массива\n";

cout << "Нажмите 4, чтобы найти количество элементов массива, которые равны среднему значению максимального и минимального элементов\n";

cout << "Нажмите 5, чтобы найти кол-во элементов меньше числа а \n";

cout << "Нажмите 6, чтобы найти кол-во элементов больше числа b\n";

cout << "Нажмите 7, чтобы узнать, есть ли ваше число в массиве \n";

cout << "Нажмите 8, чтобы поменять местами элементы массива \n";

cout << "Нажмите backspace, чтобы вернуться в меню\n";

cout << "Нажмите клавишу: ";

input = \_getch();

cout << '\n';

if (input == 8)

break;

else

{

cout << "-----Для выхода обратно в меню программы, введите backspace; кроме клавишь управления функциями-----\n";

while (1)

{

switch (input)

{

case '1':

flagarray = flagarray + 1;

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

{

A[i] = rand() % (100 - (-100)) + (-100);

}

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

{

Ansort[i] = A[i];

}

cout << "Массив создан, хотите его вывести?(y/n)\n";

char chose;

chose = \_getch();

if (chose == 'y')

{

cout << "{";

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

{

cout << A[i] << " ";

}

cout << "}\n";

}

cout << "Операция 1 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

input = \_getch();

cout << '\n';

break;

case '2':

flagsorted = flagsorted + 1;

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

quicksort(A, size - 1, 0);

cout << "Массив отсортирован, хотите его вывести?(y/n)\n";

chose = \_getch();

if (chose == 'y')

{

cout << "{";

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

{

cout << A[i] << " ";

}

cout << "}\n";

}

cout << '\n';

cout << "Операция 2 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

input = \_getch();

break;

case '3':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

if (flagsorted == 0)

{

cout << "<<Сначала отсортируйте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

maxMinSorted(A, size);

maxMinNoSorted(Ansort, size);

cout << "Операция 3 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

input = \_getch();

break;

case '4':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

if (flagsorted == 0)

{

cout << "<<Сначала отсортируйте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

averageVal(A, size);

cout << "Операция 4 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

input = \_getch();

break;

case '5':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

if (flagsorted == 0)

{

cout << "<<Сначала отсортируйте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

lessA(A, size);

cout << "Операция 5 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

input = \_getch();

break;

case '6':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

if (flagsorted == 0)

{

cout << "<<Сначала отсортируйте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

bigger(A, size);

cout << "Операция 6 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

input = \_getch();

break;

case '7':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

if (flagsorted == 0)

{

cout << "<<Сначала отсортируйте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

{

cout << "Загружается алгоритм бинарного поиска... \n";

cout << "Введите число для поиска при помощи бинарного алгоритма \n";

int number;

cin >> number;

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

binarySearch(A, size, number);

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

cout << duration\_cast<nanoseconds>(e - s).count() << " наносекунд \n";

stupidSearch(A, size);

cout << "Операция 7 --- Выполненно\n";

cout << '\n';

cin.clear();

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

case '8':

if (flagarray == 0)

{

cout << "<<Сначала создайте массив>>\n";

cout << "Нажмите клавишу: ";

input = \_getch();

break;

}

changeElements(A);

cout << "Операция 8 --- Выполненно\n";

cout << '\n';

cout << "Нажмите клавишу: ";

input = \_getch();

break;

case 8:

cout << "==========Прекращение работы подпрограммы==========\n\n";

flag = flag + 1;

break;

}

if (flag == 1)

{

flagsorted = flagsorted - 1;

flagarray = flagarray - 1;

flag = flag - 1;

break;

}

}

}

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "");

HANDLE hStdout;

COORD menuCursor;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

menuCursor.X = 0;

const int MENU\_ITEMS = 5;

int ch;

bool exit = false;

menuCursor.Y = 1;

while (!exit) {

printMenu();

SetConsoleCursorPosition(hStdout, menuCursor);

ch = \_getch();

if (ch == 224)

ch = \_getch();

switch (ch)

{

case 27: // стрелка вверх

exit = true;

break;

case 72: //стрелка вверх

menuCursor.Y--;

break;

case 80: //стрелка вниз

menuCursor.Y++;

break;

case 13: // enter

switch (menuCursor.Y) {

case 1:

menu1();

break;

case 2:

menu2();

break;

case 3:

menu3();

break;

case 4:

menu4();

break;

case 5:

exit = true;

break;

}

break;

}

//ограничивает переход

if (menuCursor.Y < 1) menuCursor.Y = 1; // не даёт уйти вверх

if (menuCursor.Y > MENU\_ITEMS) menuCursor.Y = MENU\_ITEMS; // не дает уйти вниз

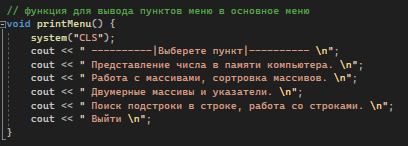
}

}

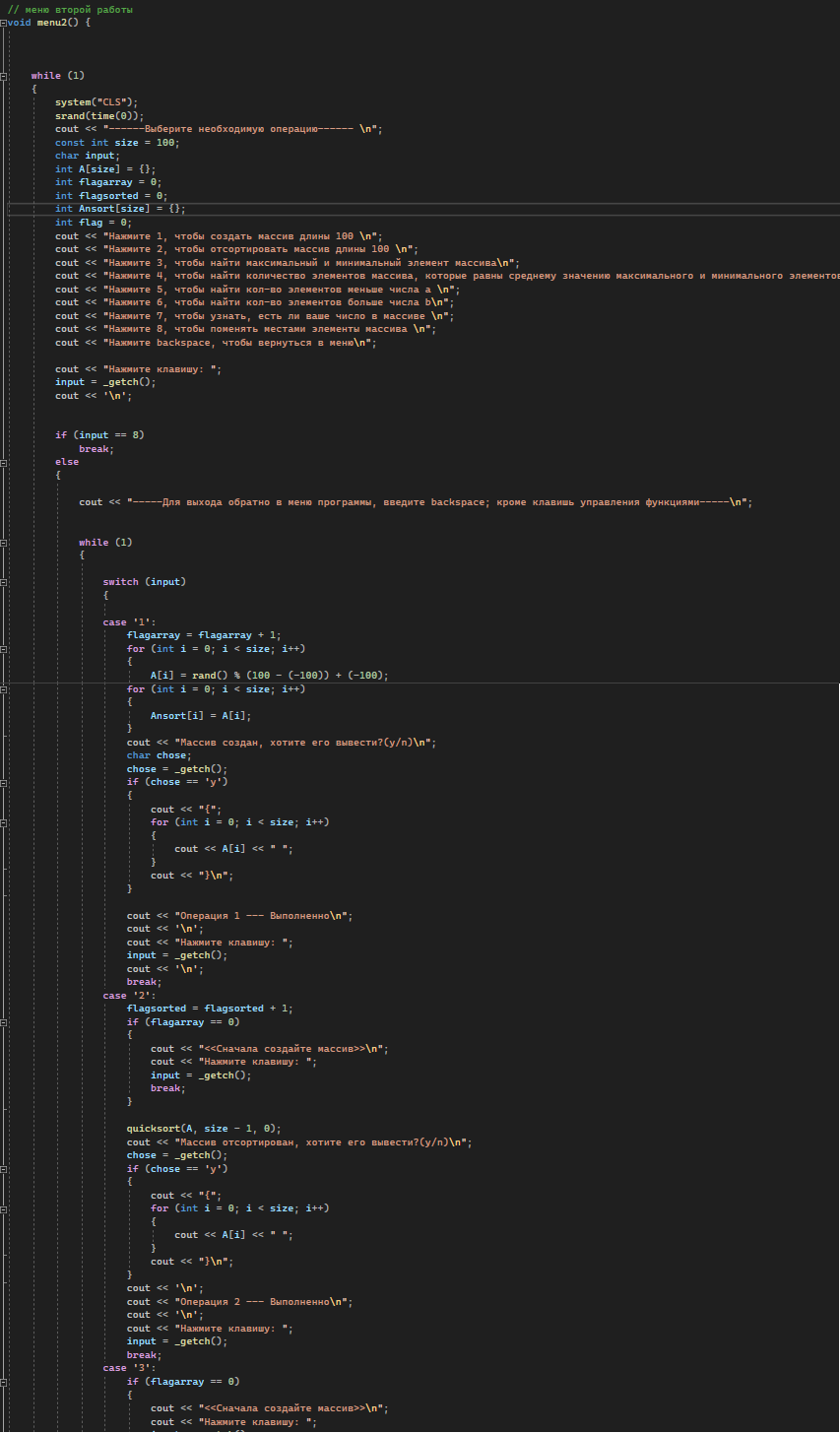
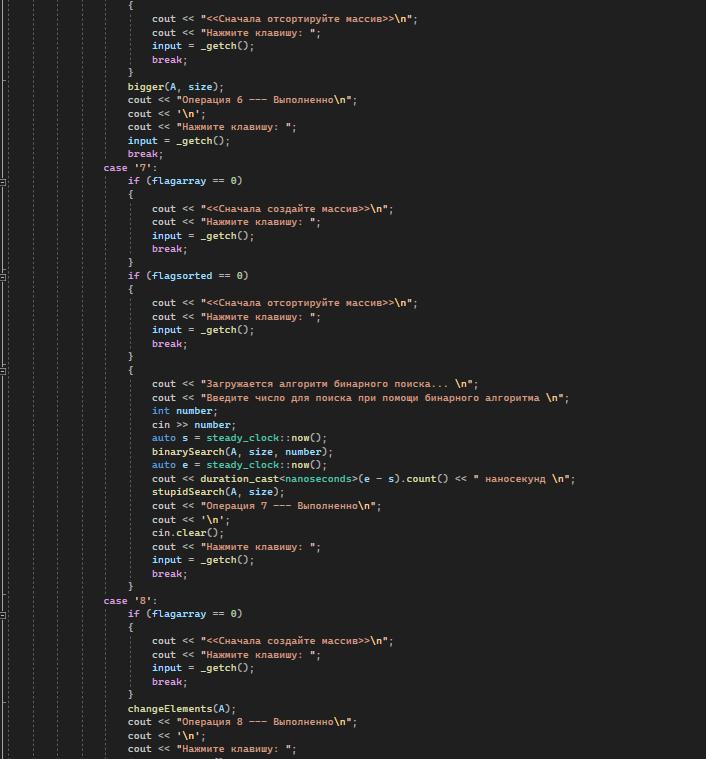
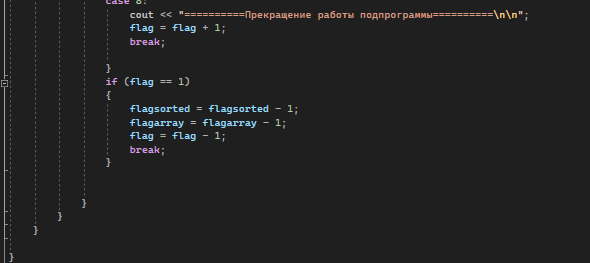
**Описание кода и использованных алгоритмов**

Работа меню:

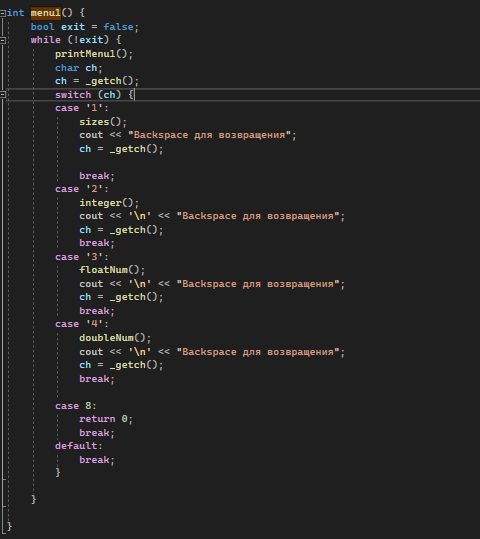




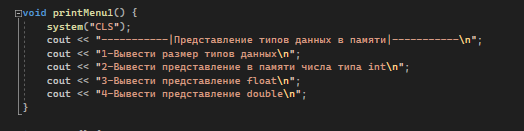
Меню второй работы:

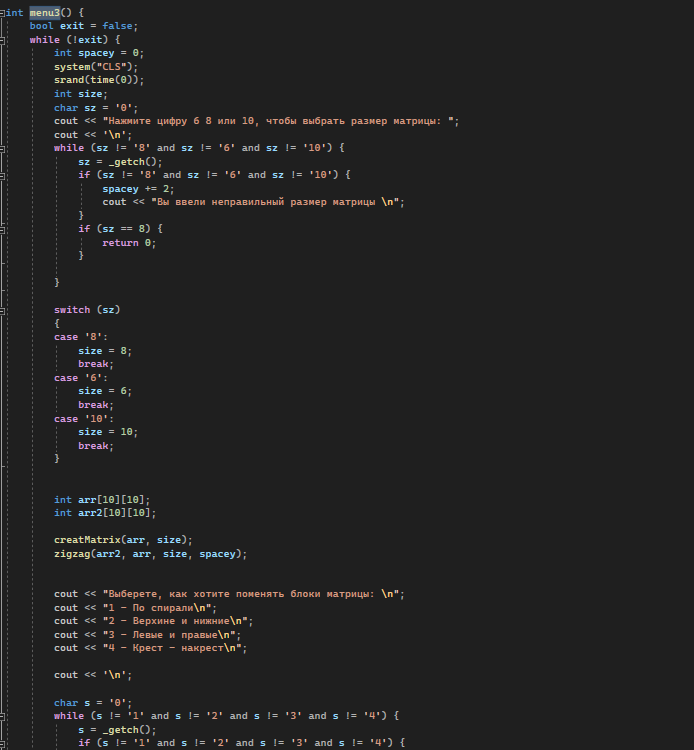
Меню первой работы:



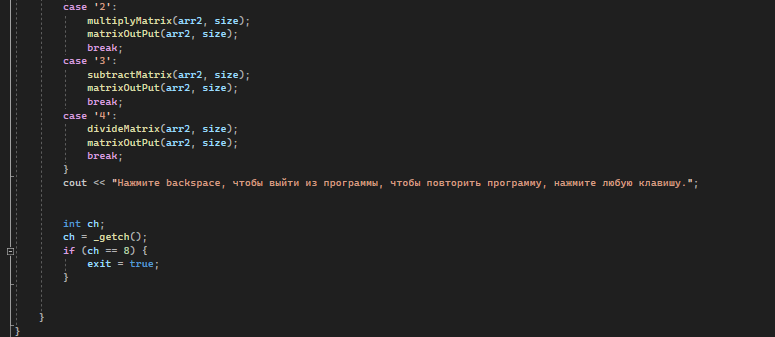
Вывод пунктов для 1го меню:



Меню 3й работы:







Меню 4 работы:

