**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

### **Тема: **Двумерные статические массивы. Указатели****.

|  |  |  |
| --- | --- | --- |
| Студент гр. 3373 | Журухин Н.А. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2023

**Цель работы.**

Научиться работать с указателями, матрицами, уметь заполнять квадратичную целочисленную матрицу порядка N по разным схемам, переставлять ее блоки. Разработать алгоритм и написать программу с помощью полученных знаний.

**Основные теоретические положения.**

Указатели и ссылки являются одними из самых важных и достаточно сложных для понимания и использования средств языка программирования. Они ориентированы на прямую работу с памятью компьютера. С помощью этих средств реализуется работа с динамической памятью и динамическими объектами, возвращение из функций измененных данных и многое другое.

Для получения адреса какого-либо программного объекта используется оператор **&**.

**Указатели** – это тоже обычные переменные, но они **служат для хранения адресов памяти**.

Указатели определяются в программе следующим образом:

**<тип данных> \*<имя переменной>**

 Здесь <**тип данных**> определяет так называемый **базовый тип указателя**.

**<Имя переменной>**является идентификатором переменной-указателя.

Признаком того, что это переменная указатель, является символ \*, располагающийся между базовым типом указателя и именем переменной-указателя. Формально указатели представляют собой обычные целые значения типа **int**и занимают в памяти 4 байта не зависимо от базового типа указателя. Значения указателей при их выводе на экран представляются как целые значения в шестнадцатеричном формате.

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. При этом указатель и переменная должны иметь один и тот же тип. Когда указателю присваивается другой указатель, то фактически первый указатель начинает также указывать на тот же адрес, на который указывает второй указатель.

Нулевой указатель (null pointer) - это указатель, который не указывает ни на какой объект. Если мы не хотим, чтобы указатель указывал на какой-то конкретный адрес, то можно присвоить ему условное нулевое значение.

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

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.

К указателям могут применяться операции сравнения >, >=, <, <=,==, !=. Операции сравнения применяются только к указателям одного типа и к значениям NULL и nullptr. Для сравнения используются номера адресов.

Иногда требуется присвоить указателю одного типа значение указателя другого типа. В этом случае следует выполнить операцию приведения типов с помощью операции “(тип\_указателя \*)”.

Для преобразования указателя к другому типу в скобках перед указателем ставится тип, к которому надо преобразовать. Причем если мы не можем просто создать объект, например, переменную типа void, то для указателя это вполне будет работать. То есть можно создать указатель типа void.

Кроме того, следует отметить, что указатель на тип char (char \*pc = &c) при выводе на консоль система интерпретирует как строку. Поэтому если мы все-таки хотим вывести на консоль адрес, который хранится в указателе типа char, то это указатель надо преобразовать к другому типу, например, к void\* или к int\*.

К указателям можно применять некоторые арифметические операции. К таким операциям относятся:  **+**,**-**, **++**, **--**. Результаты выполнения этих операций по отношению к указателям существенно отличаются от результатов соответствующих арифметических операций, выполняющихся с обычными числовыми данными.

добавление или вычитание 1 из указателя приводит к изменению его значения на размер базового типа указателя. В общем случае, например, при выполнении следующей операции:

p1 = p1 + N;  //  N – некоторое целое значение

значение указателя увеличится на **sizeof(<базовый тип указателя>) \* N**. Так, если N = 4, а p1= 100, то значение указателя **p1** увеличится на 16 и станет равно 116, и указатель будет  ссылаться на данные, расположенные по адресу 116.

Добавлять к указателям или вычитать из указателей можно только целые значения.

Поскольку упомянутые арифметические операции выполняются по-разному при их применении к указателям и обычным арифметическим типам данных, а также учитывая высший приоритет операции \*, при использовании указателей в составе выражений следует внимательно обращаться со скобками. Например, выражения

\*(p1 + 1)  и \*p1 + 1

имеют совершенно разный смысл. В первом выражении сначала изменяется адрес, а затем осуществляется обращение в память по этому измененному адресу; во втором выражении мы обращаемся по старому адресу и к значению, хранящемуся по этому адресу добавляем 1.

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

Кода мы определяем в программе некоторый массив, например,

int Arr[10];

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.

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

int Arr[10];

int \*p;

p = Arr;

Но обратное присвоение выполнить невозможно:

Arr = p;  // Ошибка

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. Нужно обратить 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 +.

**Постановка задачи.**

Разработать алгоритм и написать программу, которая:

1. Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка N (6,8,10) случайными числами от 1 до  N\*N согласно определенным схемам. Пользователь должен видеть процесс заполнения квадратичной матрицы.
2. Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии с приведенными схемами.
3. Используя арифметику указателей, сортирует элементы любой сортировкой из списка: “Shaker sort”, “Comb sort”, “Insert sort”, “Quick sort”.
4. Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

**Выполнение работы.**

Код программы представлен в приложении А.

Блок описания кода и использованных алгоритмов

Программа выполняет 4 задачи:

1. Используя арифметику указателей, заполняется квадратичная целочисленная матрица порядка N (6,8,10) случайными числами от 1 до  N\*N согласно двум определенным схемам. Пользователь видит процесс заполнения квадратичной матрицы.
2. Получается новая матрица, из матрицы п. 1, перестановкой ее блоков в соответствии с 4 приведенными схемами. Используется арифметика указателей.
3. Используя арифметику указателей, сортируются элементы любой сортировкой из списка: “Shaker sort”, “Comb sort”, “Insert sort”, “Quick sort”.
4. Уменьшаются, увеличиваются, умножаются или делятся все элементы матрицы на введенное пользователем число. Используется арифметика указателей.

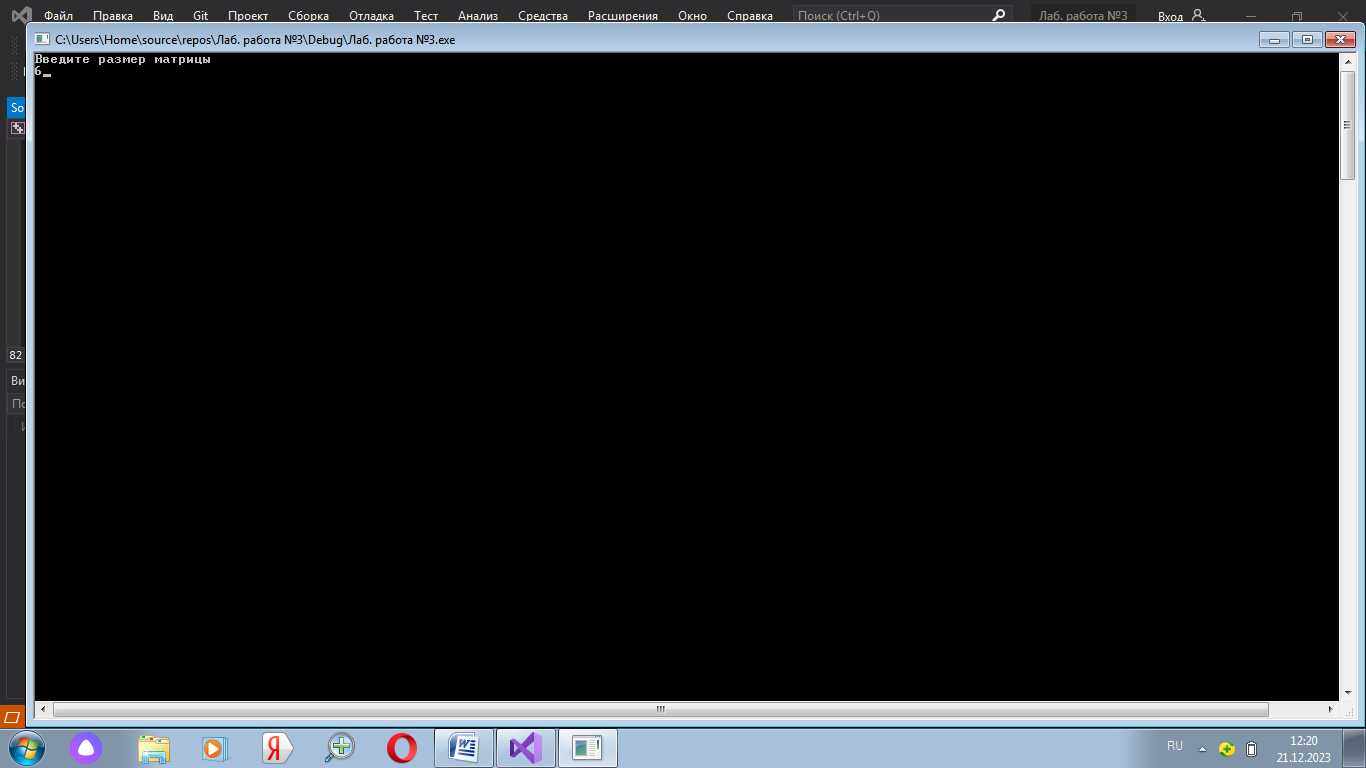
Блок скриншотов работы программы

Рис.1 Ввод размера матрицы

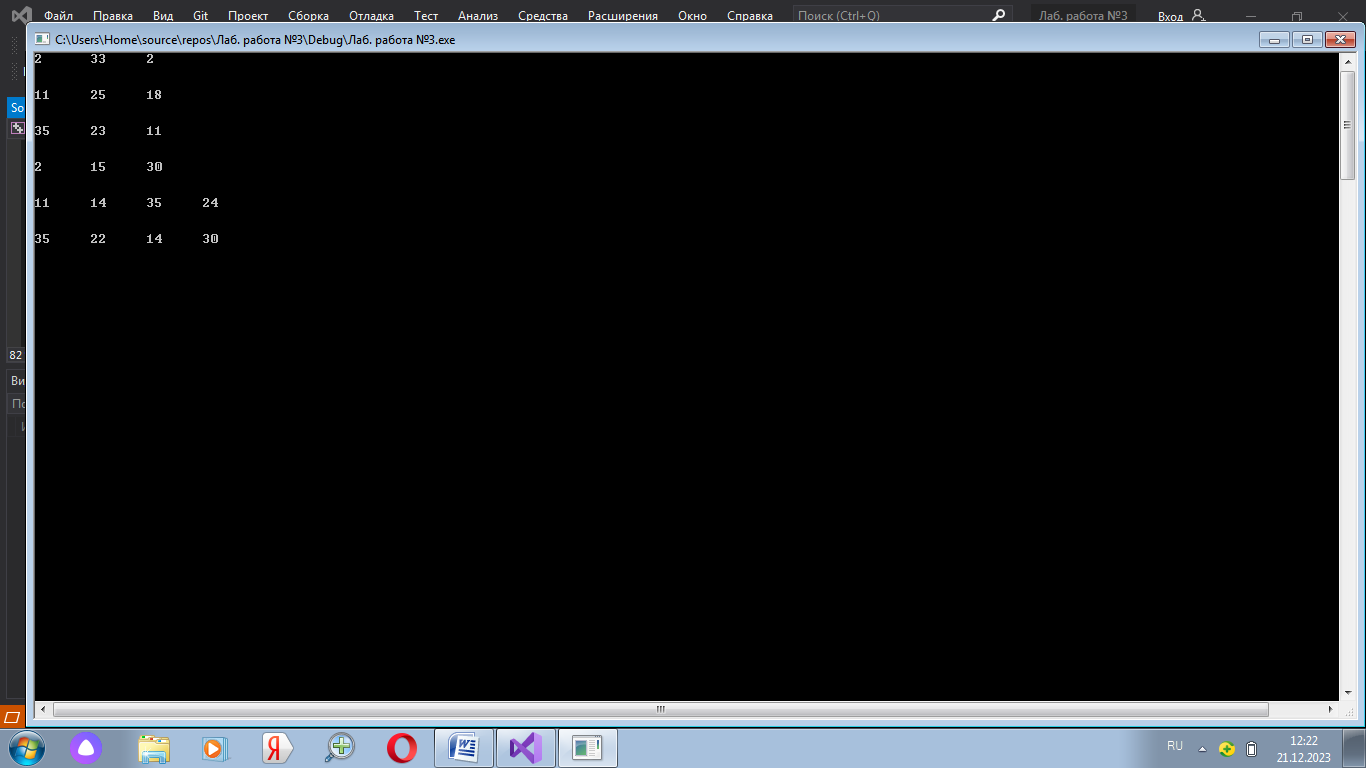


Рис.2 Заполнение матрицы по 1 образцу

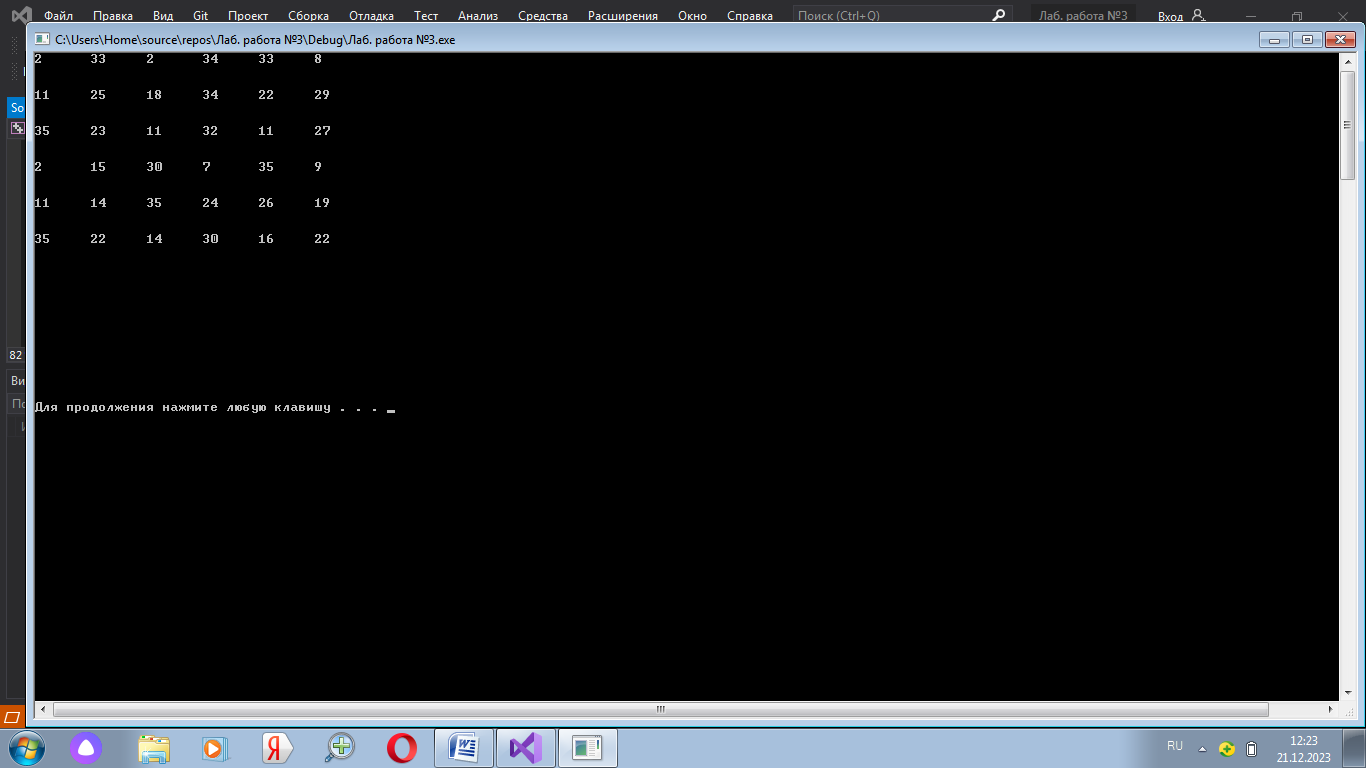
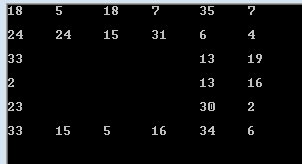


Рис.3 Заполнение матрицы по 1 образцу



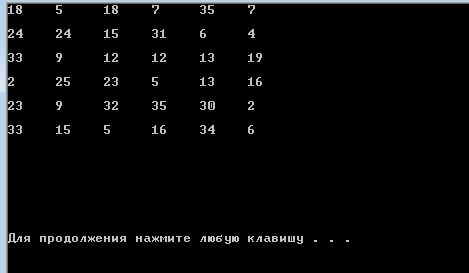
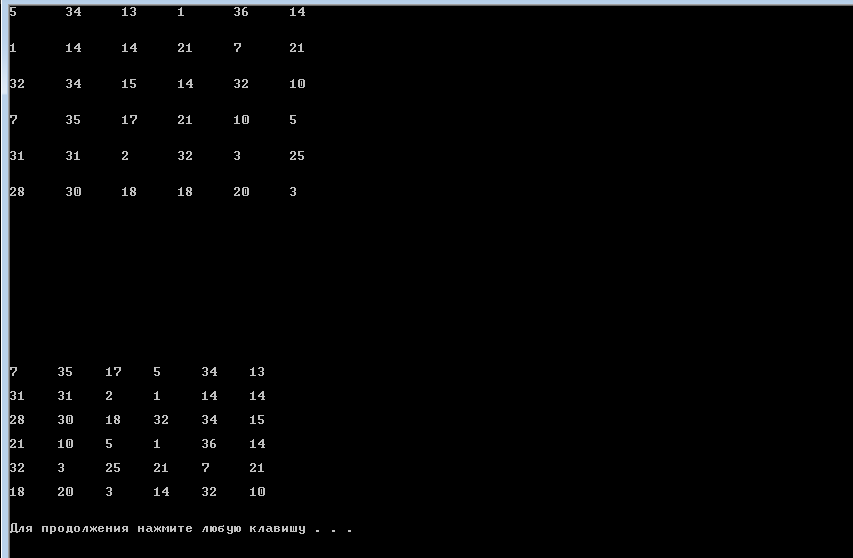


Рис.6 Перестановка блоков по образцу 1

Рис.5 Заполнение матрицы по образцу 2

Рис.4 Заполнение матрицы по образцу 2

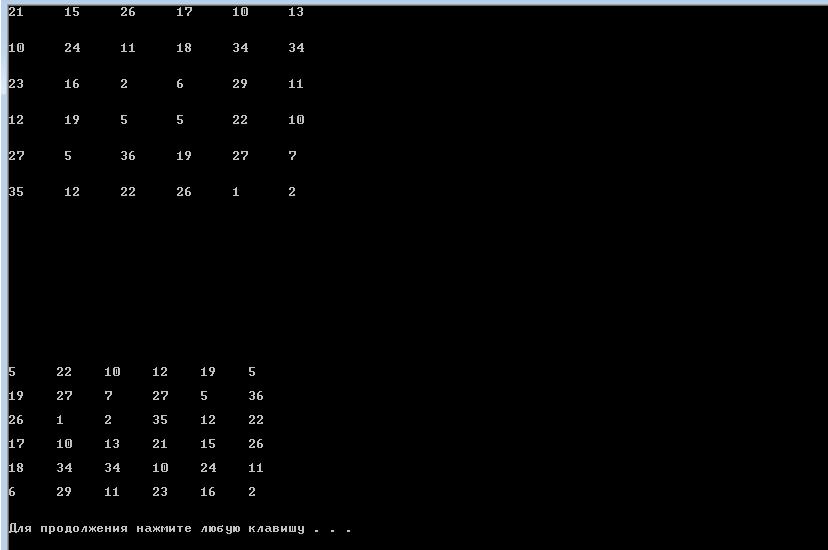


Рис.7 Перестановка блоков по образцу 2

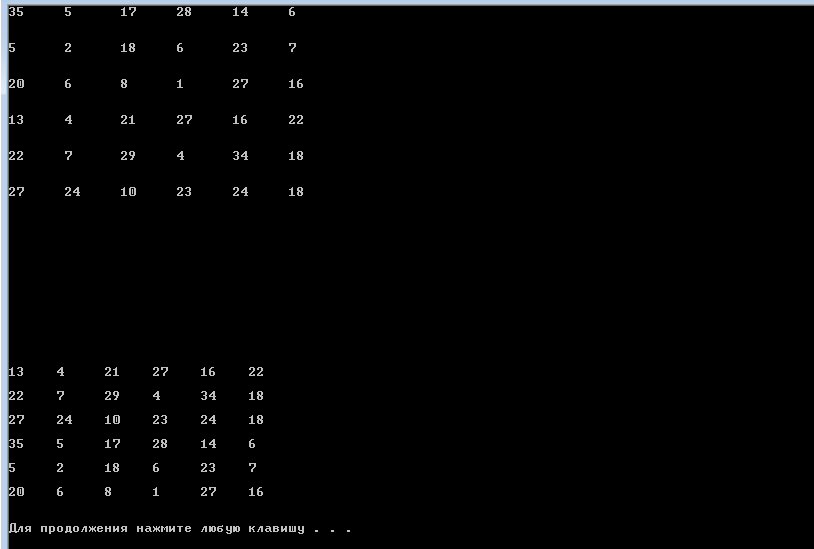


Рис.8 Перестановка блоков по образцу 3

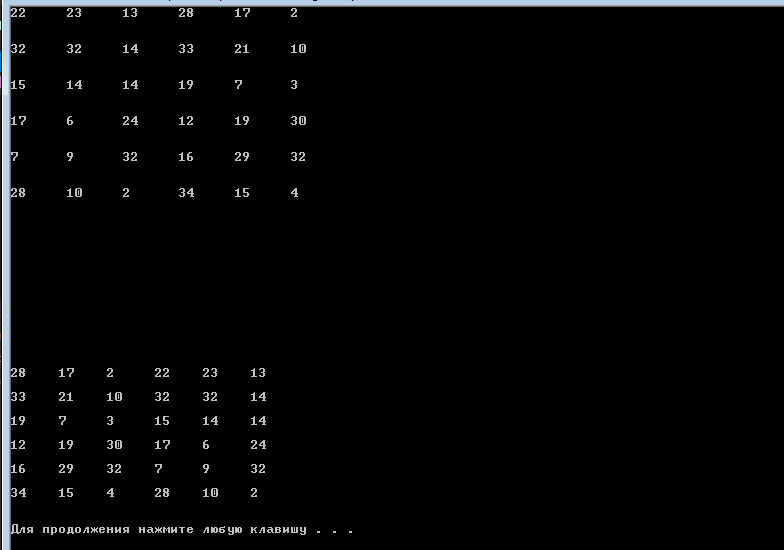


Рис.9 Перестановка блоков по образцу 4

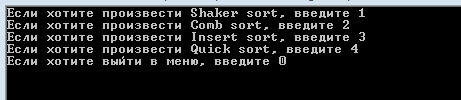


Рис.10 Использование сортировок

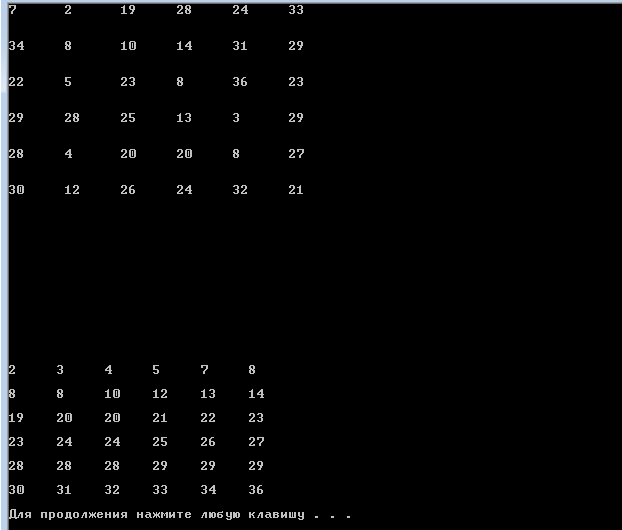


Рис. 11 Использование сортировок

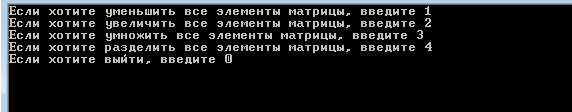


Рис. 12 Изменение элементов матрицы

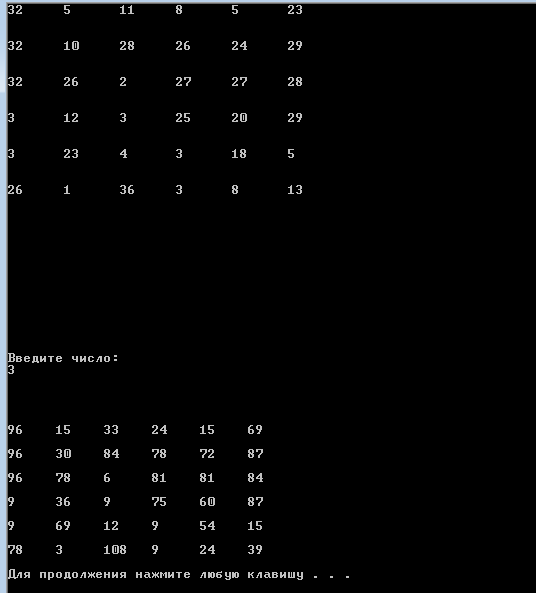
****

Рис.13 Умножение элементов матрицы

**Выводы.**

Выполнив данную лабораторную работу, я научился работать с указателями, матрицами, выводом через координаты. Были реализованы 4 вида сортировок через арифметику указателей.

Приложение А

рабочий код

#include <iostream>

#include <iomanip>

#include <time.h>

#include <Windows.h>

using namespace std;

const int N1 = 6;

const int N2 = 8;

const int N3 = 10;

void mas(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

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

int k = 0, l = i \* 6;

for (int\* next = p + i, \*end = next + N \* (N - 1); next <= end; next += N) {

\*next = rand() % (N \* N) + 1;

destCoord.X = (next - p) % N + l;

destCoord.Y = (next - p) / N + k;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next;

cout.flush();

Sleep(100);

k += 2;

}

cout << '\n';

k = (N - 1) \* 2, l = (i + 1) \* 6;

for (int\* next = p + i + 1, \*end = next + N \* (N - 1); end >= next; end -= N) {

\*end = rand() % (N \* N) + 1;

destCoord.X = (end - p) % N + l;

destCoord.Y = (end - p) / N + k;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*end;

cout.flush();

Sleep(100);

k -= 2;

}

cout << '\n';

}

cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n";

}

void spiral(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

for (int i = 0; i <= N / 2 - 1; i++) {

for (int\* next = p + i \* (N + 1), \*end = p + (N - 2) + (N - 1) \* i; next <= end; next++) {

\*next = rand() % (N \* N) + 1;

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next;

cout.flush();

Sleep(100);

}

for (int\* next = p + N - 1 + i \* (N - 1), \*end = p + N \* N - N - 1 - i \* (N + 1); next <= end; next += N) {

\*next = rand() % (N \* N) + 1;

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next;

cout.flush();

Sleep(100);

}

for (int\* next = p + N\*N - 1 - i \* (N+1), \*end = p + N \* N - 1 - (N - 2) - i \* (N - 1); next >= end; next--) {

\*next = rand() % (N \* N) + 1;

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next;

cout.flush();

Sleep(100);

}

for (int\* next = p + N \* N - 1 - (N - 1) - i \* (N - 1), \*end = p + N + i \* (N + 1); next >= end; next -= N) {

\*next = rand() % (N \* N) + 1;

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next;

cout.flush();

Sleep(100);

}

}

cout << "\n\n\n\n\n\n\n\n\n\n\n\n\n";

}

void round(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

for (int i = 0; i < N \* N / 4; i++) {

int \*num1, \*num2, \*num3, \*num4;

num1 = p + (i \* 2 / N) \* N + (i % (N / 2));

num2 = num1 + N / 2;

num4 = num1 + N \* (N / 2);

num3 = num1 + N \* (N / 2) + N / 2;

int perem1 = \*num1, perem2 = \*num2;

\*num1 = \*num4;

\*num2 = perem1;

\*num4 = \*num3;

\*num3 = perem2;

}

int \*end = p + N \* N - 1;

for (int\* next = p; next <= end; next++) {

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2 + 30;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next << " ";

if ((next - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << "\n\n";

}

void diagonal(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

for (int i = 0; i < N \* N / 4; i++) {

int\* num1, \* num2, \* num3, \* num4;

num1 = p + (i \* 2 / N) \* N + (i % (N / 2));

num2 = num1 + N / 2;

num4 = num1 + N \* (N / 2);

num3 = num1 + N \* (N / 2) + N / 2;

swap(\*num1, \*num3);

swap(\*num2, \*num4);

}

int\* end = p + N \* N - 1;

for (int\* next = p; next <= end; next++) {

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2 + 30;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next << " ";

if ((next - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << "\n\n";

}

void topBottom(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

for (int i = 0; i < N \* N / 4; i++) {

int\* num1, \* num2, \* num3, \* num4;

num1 = p + (i \* 2 / N) \* N + (i % (N / 2));

num2 = num1 + N / 2;

num4 = num1 + N \* (N / 2);

num3 = num1 + N \* (N / 2) + N / 2;

swap(\*num1, \*num4);

swap(\*num2, \*num3);

}

int \*end = p + N \* N - 1;

for (int\* next = p; next <= end; next++) {

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2 + 30;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next << " ";

if ((next - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << "\n\n";

}

void leftRight(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

for (int i = 0; i < N \* N / 4; i++) {

int\* num1, \* num2, \* num3, \* num4;

num1 = p + (i \* 2 / N) \* N + (i % (N / 2));

num2 = num1 + N / 2;

num4 = num1 + N \* (N / 2);

num3 = num1 + N \* (N / 2) + N / 2;

swap(\*num1, \*num2);

swap(\*num3, \*num4);

}

int \*end = p + N \* N - 1;

for (int\* next = p; next <= end; next++) {

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2 + 30;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next << " ";

if ((next - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << "\n\n";

}

int insert\_sort2(int \*p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

int perem, \*elem;

int \*end = p + N \* N - 1;

for (int\* next = p + 1; next <= end; next++) {

perem = \*next;

elem = next;

for (int \*next2 = next - 1; (next2 - p) >= 0 && \*next2 > perem; next2--) {

\*(next2 + 1) = \*next2;

elem = next2;

}

\*elem = perem;

}

for (int\* next2 = p; next2 <= end; next2++) {

destCoord.X = (next2 - p) % N \* 6;

destCoord.Y = (next2 - p) / N \* 2 + 30;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next2 << " ";

if ((next2 - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << '\n';

return 0;

}

int comb\_sort2(int \*p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

int swap;

float k = 1.247;

int S = N \* N - 1;

int count = 0;

while (S >= 1) {

int \*end = p + N \* N;

for (int \*next = p; next + S < end; next++) {

if (\*next > \*(next + S)) {

swap = \*(next + S);

\*(next + S) = \*next;

\*next = swap;

}

}

S /= k;

}

while (true) {

int \*end = p + N \* N - 1;

for (int \*next = p; next < end; next++) {

if (\*next > \*(next + 1)) {

swap = \*(next + 1);

\*(next + 1) = \*next;

\*next = swap;

}

else count++;

}

if (count == N \* N - 1)

break;

else

count = 0;

}

int\* end = p + N \* N - 1;

for (int\* next2 = p; next2 <= end; next2++) {

destCoord.X = (next2 - p) % N \* 6;

destCoord.Y = (next2 - p) / N \* 2 + 30;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next2 << " ";

if ((next2 - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << '\n';

return 0;

}

int shaker\_sort2(int \*p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

bool flag;

for (int\* next = p, \*end = p + N \* N / 2; next < end; next++) {

flag = false;

for (int \*next2 = next, \*end2 = p + N \* N - 1 - (next - p); next2 < end2; next2++) {

if (\*next2 > \*(next2 + 1)) {

swap(\*next2, \*(next2 + 1));

flag = true;

}

}

for (int \*next3 = p + N \* N - (next - p) - 2; next3 >= next; next3--) {

if (\*next3 < \*(next3 - 1)) {

swap(\*next3, \*(next3 - 1));

flag = true;

}

}

if (flag == false) {

break;

}

}

int\* end = p + N \* N - 1;

for (int\* next2 = p; next2 <= end; next2++) {

destCoord.X = (next2 - p) % N \* 6;

destCoord.Y = (next2 - p) / N \* 2 + 30;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next2 << " ";

if ((next2 - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << '\n';

return 0;

}

void quicksort2(int\* p, int end, int begin) {

int mid;

int f = begin;

int l = end;

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

while (f < l)

{

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

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

if (f <= l)

{

swap(\*(p + f), \*(p + l));

f++;

l--;

}

}

if (begin < l) quicksort2(p, l, begin);

if (f < end) quicksort2(p, end, f);

}

void matPlus(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

int num;

cin >> num;

int \*end = p + N \* N - 1;

for (int\* next = p; next <= end; next++) {

\*next += num;

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2 + 35;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next << " ";

if ((next - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << '\n';

}

void matMinus(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

int num;

cin >> num;

int\* end = p + N \* N - 1;

for (int\* next = p; next <= end; next++) {

\*next -= num;

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2 + 35;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next << " ";

if ((next - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << '\n';

}

void matMultiplication(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

int num;

cin >> num;

int\* end = p + N \* N - 1;

for (int\* next = p; next <= end; next++) {

\*next \*= num;

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2 + 35;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next << " ";

if ((next - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << '\n';

}

void matDivision(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

int num;

cin >> num;

if (num) {

int \*end = p + N \* N - 1;

for (int\* next = p; next <= end; next++) {

\*next /= num;

destCoord.X = (next - p) % N \* 6;

destCoord.Y = (next - p) / N \* 2 + 35;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next << " ";

if ((next - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

}

else {

cout << "Ошибка, деление на 0";

}

cout << '\n';

}

void idz(int \* ip1, int \*ip2, int \*ip3, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

int\* end1;

end1 = ip1 + N \* N - 1;

for (int\* next1 = ip1; next1 <= end1; next1++) {

\*next1 = rand() % (N \* N) + 1;

destCoord.X = (next1 - ip1) % N \* 6;

destCoord.Y = (next1 - ip1) / N \* 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next1 << " ";

if ((next1 - ip1 + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << '\n';

int\* end2;

end2 = ip2 + N \* N - 1;

for (int\* next2 = ip2; next2 <= end2; next2++) {

\*next2 = rand() % (N \* N) + 1;

destCoord.X = (next2 - ip2) % N \* 6;

destCoord.Y = (next2 - ip2) / N \* 2 + 25;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next2 << " ";

if ((next2 - ip2 + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << '\n';

int\* end3;

end3 = ip3 + N \* N - 1;

for (int\* next3 = ip3; next3 <= end3; next3++) {

\*next3 = (\*(ip1 + (next3 - ip3))) + (\*(ip2 + (next3 - ip3)));

destCoord.X = (next3 - ip3) % N \* 6;

destCoord.Y = (next3 - ip3) / N \* 2 + 50;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next3 << " ";

if ((next3 - ip3 + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << '\n';

}

void mSort(int\* p, int N) {

HANDLE hStdout;

COORD destCoord;

hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

cout << "Если хотите произвести Shaker sort, введите 1\n";

cout << "Если хотите произвести Comb sort, введите 2\n";

cout << "Если хотите произвести Insert sort, введите 3\n";

cout << "Если хотите произвести Quick sort, введите 4\n";

cout << "Если хотите выйти в меню, введите 0\n";

int perem;

cin >> perem;

switch (perem) {

case 1:

system("cls");

mas(p, N);

shaker\_sort2(p, N);

break;

case 2:

system("cls");

mas(p, N);

comb\_sort2(p, N);

break;

case 3:

system("cls");

mas(p, N);

insert\_sort2(p, N);

break;

case 4:

system("cls");

mas(p, N);

quicksort2(p, N \* N - 1, 0);

for (int\* next2 = p, \*end = p + N \* N - 1; next2 <= end; next2++) {

destCoord.X = (next2 - p) % N \* 6;

destCoord.Y = (next2 - p) / N \* 2 + 30;

SetConsoleCursorPosition(hStdout, destCoord);

cout << \*next2 << " ";

if ((next2 - p + 1) % N == 0) {

cout << "\n";

}

cout.flush();

Sleep(100);

}

cout << '\n';

break;

case 0:

break;

default:

cout << "Введено неправильное число\n";

break;

}

}

void action(int\* p, int N) {

cout << "Если хотите уменьшить все элементы матрицы, введите 1\n";

cout << "Если хотите увеличить все элементы матрицы, введите 2\n";

cout << "Если хотите умножить все элементы матрицы, введите 3\n";

cout << "Если хотите разделить все элементы матрицы, введите 4\n";

cout << "Если хотите выйти, введите 0\n";

int perem;

cin >> perem;

switch (perem) {

case 1:

system("cls");

mas(p, N);

cout << "Введите число:\n";

matMinus(p, N);

break;

case 2:

system("cls");

mas(p, N);

cout << "Введите число:\n";

matPlus(p, N);

break;

case 3:

system("cls");

mas(p, N);

cout << "Введите число:\n";

matMultiplication(p, N);

break;

case 4:

system("cls");

mas(p, N);

cout << "Введите число:\n";

matDivision(p, N);

break;

case 0:

break;

default:

cout << "Введено неправильное число\n";

break;

}

}

int main();

void menu(int \*p, int N, int \*ip1, int \*ip2, int \*ip3) {

cout << "Если хотите заполнить квадратичную матрицу змейкой, введите 1\n";

cout << "Если хотите заполнить квадратичную матрицу по спирали, введите 2\n";

cout << "Если хотите получить новую матрицу через изменения по кругу, введите 3\n";

cout << "Если хотите получить новую матрицу через изменения по диагоналям, введите 4\n";

cout << "Если хотите получить новую матрицу через изменения по вертикали, введите 5\n";

cout << "Если хотите получить новую матрицу через изменения по горизонтали, введите 6\n";

cout << "Если хотите использовать сортировки для матрицы, введите 7\n";

cout << "Если хотите изменить матрицу через введенное число, введите 8\n";

cout << "Если хотите вызвать ИДЗ, введите 9\n";

cout << "Если хотите изменить размер матрицы, введите 10\n";

cout << "Если хотите выйти, введите 0\n";

int num;

cin >> num;

switch (num) {

case 1:

system("cls");

mas(p, N);

system("Pause");

system("cls");

menu(p, N, ip1, ip2, ip3);

break;

case 2:

system("cls");

spiral(p, N);

system("Pause");

system("cls");

menu(p, N, ip1, ip2, ip3);

break;

case 3:

system("cls");

mas(p, N);

round(p, N);

system("Pause");

system("cls");

menu(p, N, ip1, ip2, ip3);

break;

case 4:

system("cls");

mas(p, N);

diagonal(p, N);

system("Pause");

system("cls");

menu(p, N, ip1, ip2, ip3);

break;

case 5:

system("cls");

mas(p, N);

topBottom(p, N);

system("Pause");

system("cls");

menu(p, N, ip1, ip2, ip3);

break;

case 6:

system("cls");

mas(p, N);

leftRight(p, N);

system("Pause");

system("cls");

menu(p, N, ip1, ip2, ip3);

break;

case 7:

system("cls");

mSort(p, N);

system("Pause");

system("cls");

menu(p, N, ip1, ip2, ip3);

break;

case 8:

system("cls");

action(p, N);

system("Pause");

system("cls");

menu(p, N, ip1, ip2, ip3);

break;

case 9:

system("cls");

idz(ip1, ip2, ip3, N);

system("Pause");

system("cls");

menu(p, N, ip1, ip2, ip3);

break;

case 10:

system("cls");

main();

case 0:

exit(0);

default:

cout << "Введено неправильное число\n";

system("Pause");

system("cls");

menu(p, N, ip1, ip2, ip3);

break;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int matrics1[N1][N1], matrics2[N2][N2], matrics3[N3][N3];

int\* p1 = \*matrics1, \* p2 = \*matrics2, \* p3 = \*matrics3;

int idmat1[N1][N1], idmat2[N1][N1], idmat3[N1][N1];

int\* ip1 = \*idmat1, \* ip2 = \*idmat2, \* ip3 = \*idmat3;

int idmat4[N2][N2], idmat5[N2][N2], idmat6[N2][N2];

int\* ip4 = \*idmat4, \* ip5 = \*idmat5, \* ip6 = \*idmat6;

int idmat7[N3][N3], idmat8[N3][N3], idmat9[N3][N3];

int\* ip7 = \*idmat7, \* ip8 = \*idmat8, \* ip9 = \*idmat9;

cout << "Введите размер матрицы\n";

int num;

cin >> num;

switch (num) {

case 6:

system("cls");

menu(p1, N1, ip1, ip2, ip3);

break;

case 8:

system("cls");

menu(p2, N2, ip4, ip5, ip6);

break;

case 10:

system("cls");

menu(p3, N3, ip7, ip8, ip9);

break;

default:

cout << "Введено неправильное число\n";

system("Pause");

system("cls");

main();

break;

}

return 0;

}