**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. 4373 |  | Шепелев Д.Н. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

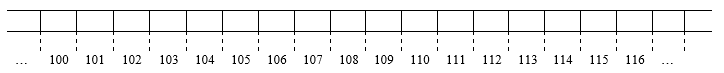
**Цель работы.**

Изучение и организация массивов данных; получение практических навыков работы со массивами; изучение сортировок.

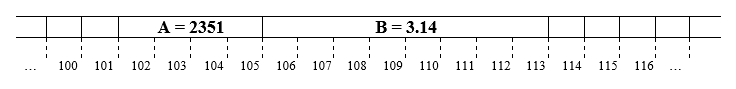
**Основные теоретические положения.**

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. К использованию указателей и ссылок мы будем неоднократно возвращаться в последующих разделах.

Все данные (переменные, константы и др.) хранятся в памяти. Память представляет собой непрерывную последовательность ячеек (байтов), каждая из которых имеет свой номер – адрес:



При определении, например, некоторой переменной, она располагается в памяти по определенному адресу и занимает столько ячеек, сколько требует тип этой переменной. Пусть, например, имеется переменные **int A = 2351** и **double B = 3.1** и пусть они располагаются в памяти так:



Говорят, что переменная **А** располагается по адресу 102 и занимает 4 байта, а переменная **B** имеет адрес 106 и занимает 8 байт памяти.

Для получения адреса какого-либо программного объекта используется оператор **&**. Например, если выполнить фрагмент следующей программы (в предположении, что переменные A и B располагаются в памяти, как это показано на предыдущем рисунке):

int A = 2351;

double B = 3.14;

cout <<  “Значение переменной А: ” << A << endl;

cout <<  “Адрес переменной А: ” << &A << endl;

cout <<  “Значение переменной В: ” << В << endl;

cout <<  “Адрес переменной В: ” << &В << endl;

получим следующий результат:

Значение переменной А: 2351

Адрес переменной А: 102

Значение переменной В: 3.14

Адрес переменной В: 106

Правда, значения адресов переменных будут выведены в шестнадцатеричном  формате.

**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.

Например:

int \*p1;

double \*p2;

Здесь определены две переменные-указатели (или просто – два указателя). Указатель **p1** является переменной-указателем на базовый тип **int**(или, как говорят, переменная **p1** указывает  на **int**- значение), а указатель **p2** указывает на **double**– значение.

Иными словами, переменная **p1** предназначена для хранения адресов участков памяти, размер которых соответствует типу **int**(4 байта), а переменная **p2** - для хранения адресов участков памяти, размер которых соответствует типу **double**(8 байт).

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.

**Постановка задачи.**

1)    Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка *N* (6,8,10) случайными числами от 1 до  N\*N согласно схемам, приведенным на рисунках. Пользователь должен видеть процесс заполнения квадратичной матрицы (\*Для манипуляции с элементами используйте только арифметику указателей):



2)    Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами (\*Для манипуляции с элементами используйте только арифметику указателей):



3)    Используя арифметику указателей, сортирует элементы любой сортировкой из списка ниже (если во 2 ПР вы реализовывали одну из представленных сортировок, рекомендуется переиспользовать написанный код и модернизировать его для работы с указатями).   
Варианты сортировок:

Shaker sort;

Comb sort;

Insert sort;

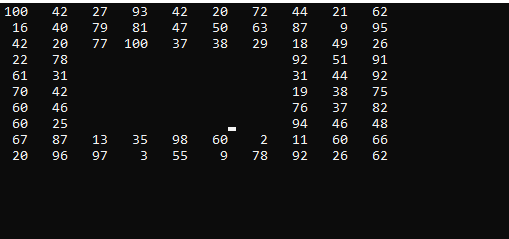
Quick sort;

4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число (\*Для манипуляции с элементами используйте только арифметику указателей).

**Выполнение работы.**

Код программы представлен в приложении А.

Пример работы кода :



**Выводы.**

Я изучил массивы, получил практические навыки работы с ними и определил преимущества и недостатки разных сортировок.

Приложение А

#include <iostream>

#include <iomanip>

#include <windows.h>

#include <stdio.h>

#include <conio.h>

using namespace std;

HANDLE hStdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

COORD destCoord;

void printNumber(int number, int x, int y) {

destCoord.X = x;

destCoord.Y = y;

SetConsoleCursorPosition(hStdout, destCoord);

cout << setw(5) << number << '\r';

cout.flush();

}

void printEl(int number, char colour, int\*\* pattern, int rows, int colums) {

int el, x, y;

switch (colour) {

case 'r':

SetConsoleTextAttribute(hStdout, FOREGROUND\_RED | FOREGROUND\_INTENSITY);

break;

case 'g':

SetConsoleTextAttribute(hStdout, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

break;

case 'b':

SetConsoleTextAttribute(hStdout, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

break;

case 'y':

SetConsoleTextAttribute(hStdout, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

break;

case 'v':

SetConsoleTextAttribute(hStdout, FOREGROUND\_RED | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

break;

}

el = pattern[number] - pattern[0];

x = (el % colums) \* 5;

y = (el + 1 == colums \* rows ? rows - 1 : el / colums);

printNumber(\*pattern[number], x, y);

SetConsoleTextAttribute(hStdout, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

}

void printMat(int\*\* mat, int rows, int colums, int x, int y) {

//system("cls");

int\* turtle = &mat[0][0];

for (int i = 1; i <= rows \* colums; ++i, ++turtle, x+=5) {

printNumber(\*turtle, x, y);

if (i % colums == 0){

++y;

x -=5\* colums;

}

}

}

void fillMatSpiral(int\*\* mat, int rows, int colums, int speed) {

system("cls");

int\* turtle = &mat[0][0];

int k = 1, c = 1 , x = 0, y = 0;

int f = 1, l = 1;

bool flag = true;

char side = 'r';

srand(time(0));

while (c <= rows \* colums){

switch (side) {

case 'r':

k = 1;

side = 'd';

break;

case 'd':

k = colums;

side = 'l';

break;

case 'l':

k = -1;

side = 'u';

break;

case 'u':

k = -colums;

side = 'r';

break;

}

if (side == 'd' || side == 'u') {

for (int i = 0; colums - l == 0 ? i <= colums - l : i < colums - l ; ++i, c++, side == 'd' ? x += 5 : x -= 5, turtle += k) {

\*turtle = rand() % (rows\*colums) + 1;

printNumber(\*turtle, x, y);

Sleep(speed);

}

flag ? flag = false : l += 1;

}

else {

for (int i = 0; rows - f == 0? i <= rows - f : i < rows - f; ++i, c++, side == 'l' ? y += 1 : y -= 1, turtle += k) {

\*turtle = rand() % (rows\*colums) + 1;

printNumber(\*turtle, x, y);

Sleep(speed);

}

f += 1;

}

}

destCoord.X = 0;

destCoord.Y = rows+1;

SetConsoleCursorPosition(hStdout, destCoord);

}

void fillMatVert(int\*\* mat, int rows, int colums, int speed) {

system("cls");

int\* turtle = &mat[0][0];

int k = 1, c = 1, x = 0, y = 0;

char side = 'd';

srand(time(0));

while (c <= rows \* colums) {

switch (side) {

case 'd':

k = colums;

side = 'u';

break;

case 'u':

k = -colums;

side = 'd';

break;

}

for (int i = 0; i < rows; ++i, ++c, turtle += k) {

\*turtle = rand() % (rows\*colums) + 1;

printNumber(\*turtle, x, y);

Sleep(speed);

side == 'u' ? y += 1 : y -= 1;

}

turtle++;

x+=5;

side == 'u' ? y -= 1 : y += 1;

turtle-=k;

}

destCoord.X = 0;

destCoord.Y = rows + 1;

SetConsoleCursorPosition(hStdout, destCoord);

}

void fillMatSimple(int\*\* mat, int rows, int colums, int speed) {

system("cls");

int\* turtle = &mat[0][0];

int c = 0, x = 0, y = 0;

srand(time(0));

for (int i = 0; i < rows\*colums; ++i, turtle++, c++) {

\*turtle = rand() % (rows \* colums) + 1;

x = (c % colums) \* 5;

y = (c + 1 == colums \* rows ? rows - 1 : c / colums);

printNumber(\*turtle, x, y);

Sleep(speed);

}

destCoord.X = 0;

destCoord.Y = rows + 1;

SetConsoleCursorPosition(hStdout, destCoord);

}

void replaceBlocksRound(int\*\* mat, int rows, int colums) {

int\* turtle = &mat[0][0];

int\* turtle2;

int c = 0;

int rowsParity, columsParity;

rows % 2 == 0 ? rowsParity = 0 : rowsParity = 1;

colums % 2 == 0 ? columsParity = 0 : columsParity = 1;

for (int i = 1; i <= (colums / 2) \* (rows / 2); ++i) {

turtle2 = turtle + (colums / 2 + columsParity);

swap(\*turtle, \*turtle2);

turtle2 += (rows / 2 + rowsParity) \* rows;

swap(\*turtle, \*turtle2);

turtle2 = turtle + (rows / 2 + rowsParity) \* rows;

swap(\*turtle, \*turtle2);

i % (colums / 2) == 0 ? turtle = &mat[0][0] + colums \* (i / (colums / 2)) : turtle++;

}

}

void replaceBlocksObliq(int\*\* mat, int rows, int colums) {

int\* turtle = &mat[0][0];

int\* turtle2;

int rowsParity, columsParity;

rows % 2 == 0 ? rowsParity = 0 : rowsParity = 1;

colums % 2 == 0 ? columsParity = 0 : columsParity = 1;

for (int i = 1; i <= (colums / 2) \* (rows / 2); ++i) {

turtle2 = turtle + (colums / 2 + columsParity) + (rows / 2 + rowsParity) \* rows;

swap(\*turtle, \*turtle2);

i % (colums / 2) == 0 ? turtle = &mat[0][0] + colums \* (i / (colums / 2)) : turtle++;

}

turtle = &mat[rows / 2 + rowsParity][0];

for (int i = 1; i <= (colums / 2) \* (rows / 2); ++i) {

turtle2 = turtle + (colums / 2 + columsParity) - (rows / 2 + rowsParity) \* rows;

swap(\*turtle, \*turtle2);

i % (colums / 2) == 0 ? turtle = &mat[rows / 2 + rowsParity][0] + colums \* (i / (colums / 2)) : turtle++;

}

}

void replaceBlocksVert(int\*\* mat, int rows, int colums) {

int\* turtle = &mat[0][0];

int\* turtle2;

int rowsParity, columsParity;

rows % 2 == 0 ? rowsParity = 0 : rowsParity = 1;

colums % 2 == 0 ? columsParity = 0 : columsParity = 1;

for (int i = 1; i <= (colums / 2) \* (rows / 2); ++i) {

turtle2 = turtle + (rows / 2 + rowsParity) \* rows;

swap(\*turtle, \*turtle2);

i % (colums / 2) == 0 ? turtle = &mat[0][0] + colums \* (i / (colums / 2)) : turtle++;

}

turtle = &mat[0][colums / 2 + columsParity];

for (int i = 1; i <= (colums / 2) \* (rows / 2); ++i) {

turtle2 = turtle + (rows / 2 + rowsParity) \* rows;

swap(\*turtle, \*turtle2);

i % (colums / 2) == 0 ? turtle = &mat[0][colums / 2 + columsParity] + colums \* (i / (colums / 2)) : turtle++;

}

}

void replaceBlocksHor(int\*\* mat, int rows, int colums) {

int\* turtle = &mat[0][0];

int\* turtle2;

int rowsParity, columsParity;

rows % 2 == 0 ? rowsParity = 0 : rowsParity = 1;

colums % 2 == 0 ? columsParity = 0 : columsParity = 1;

for (int i = 1; i <= (colums / 2) \* (rows / 2); ++i) {

turtle2 = turtle + (colums / 2 + columsParity);

swap(\*turtle, \*turtle2);

i % (colums / 2) == 0 ? turtle = &mat[0][0] + colums \* (i / (colums / 2)) : turtle++;

}

turtle = &mat[rows / 2 + rowsParity][0];

for (int i = 1; i <= (colums / 2) \* (rows / 2); ++i) {

turtle2 = turtle + (colums / 2 + columsParity);

swap(\*turtle, \*turtle2);

i % (colums / 2) == 0 ? turtle = &mat[rows / 2 + rowsParity][0] + colums \* (i / (colums / 2)) : turtle++;

}

}

void arithOperMat(int\*\* mat, int rows, int colums, char operation, int number) {

int\* turtle = &mat[0][0];

for (int i = 0; i < rows \* colums; ++i, ++turtle) {

switch (operation) {

case '+':

\*turtle += number;

break;

case '-':

\*turtle -= number;

break;

case '\*':

\*turtle \*= number;

break;

case '/':

\*turtle /= number;

break;

}

}

}

void createSpiralPointersPattern(int\*\* pattern, int\*\* mat, int rows, int colums) {

int\* turtle = &mat[0][0];

int k = 1, c = 0;

int f = 1, l = 1;

bool flag = true;

char side = 'r';

while (c < rows \* colums) {

switch (side) {

case 'r':

k = 1;

side = 'd';

break;

case 'd':

k = colums;

side = 'l';

break;

case 'l':

k = -1;

side = 'u';

break;

case 'u':

k = -colums;

side = 'r';

break;

}

if (side == 'd' || side == 'u') {

for (int i = 0; colums - l == 0 ? i <= colums - l : i < colums - l; ++i, c++, turtle += k) {

pattern[c] = turtle;

}

flag ? flag = false : l += 1;

}

else {

for (int i = 0; rows - f == 0 ? i <= rows - f : i < rows - f; ++i, c++, turtle += k) {

pattern[c] = turtle;

}

f += 1;

}

}

}

void createVertPointersPattern(int\*\* pattern, int\*\* mat, int rows, int colums) {

int\* turtle = &mat[0][0];

int k, c = 1;

char side = 'd';

while (c < rows \* colums) {

switch (side) {

case 'd':

k = colums;

side = 'u';

break;

case 'u':

k = -colums;

side = 'd';

break;

}

for (int i = 0; i < rows; ++i, ++c, turtle += k) {

pattern[c - 1] = turtle;

}

turtle++;

turtle -= k;

}

}

void createVertPointersObliq(int\*\* pattern, int\*\* mat, int rows, int colums) {

int\* turtle = &mat[0][0];

int k, ln = 1 , lp = 0, c = 0;

char side = 'd';

while (c + 1 < rows \* colums) {

switch (side) {

case 'd':

k = colums - 1;

while (ln > lp) {

pattern[c] = turtle;

turtle += k;

c++;

}

turtle -= k;

turtle += colums;

if (turtle < &mat[0][0] || turtle > &mat[rows - 1][colums - 1]) {

turtle -= colums;

turtle++;

}

side = 'u';

break;

case 'u':

k = -colums + 1;

while (turtle >= &mat[0][0] && turtle <= &mat[rows - 1][colums - 1]) {

pattern[c] = turtle;

turtle += k;

c++;

}

turtle -= k;

turtle++;

if (turtle < &mat[0][0] || turtle > &mat[rows - 1][colums - 1]) {

turtle--;

turtle += colums;

}

side = 'd';

break;

}

}

}

void createVertPointersSimple(int\*\* pattern, int\*\* mat, int rows, int colums) {

int\* turtle = &mat[0][0];

for (int i = 0; i < rows \* colums; ++i, turtle++) {

pattern[i] = turtle;

}

}

void bubbleSort(int\*\* mat, int\*\* pattern, int rows, int colums, int speed) {

system("cls");

printMat(mat, rows, colums, 0, 0);

int x, y, el;

bool isChanged = true;

int\*\* end = &pattern[rows \* colums - 1];

while (isChanged == true) {

isChanged = false;

for (int\*\* i = pattern; i < end; ++i) {

printEl(i - pattern, 'r', pattern, rows, colums);

printEl((i - pattern) + 1, 'y', pattern, rows, colums);

Sleep(speed/2);

if (\*\*(i + 1) < \*\*i) {

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

isChanged = true;

printEl(i - pattern, 'y', pattern, rows, colums);

printEl((i - pattern) + 1, 'r', pattern, rows, colums);

Sleep(speed/2);

}

printEl(i - pattern, 'w', pattern, rows, colums);

printEl((i - pattern) + 1, 'r', pattern, rows, colums);

Sleep(speed/2);

}

printEl((end - pattern), 'g', pattern, rows, colums);

--end;

}

SetConsoleTextAttribute(hStdout, FOREGROUND\_GREEN);

printMat(mat, rows, colums, 0, 0);

Sleep(speed);

SetConsoleTextAttribute(hStdout, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

}

void shakerSort(int\*\* mat, int\*\* pattern, int rows, int colums, int speed) {

system("cls");

printMat(mat, rows, colums, 0, 0);

bool isChanged = true;

int x, y, el;

int\*\* start = pattern, \*\* end = &pattern[rows \* colums - 1];

while (isChanged == true) {

isChanged = false;

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

printEl(i - pattern, 'r', pattern, rows, colums);

printEl((i - pattern) + 1, 'y', pattern, rows, colums);

Sleep(speed / 2);

if (\*\*(i + 1) < \*\*i) {

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

isChanged = true;

printEl(i - pattern, 'y', pattern, rows, colums);

printEl((i - pattern) + 1, 'r', pattern, rows, colums);

Sleep(speed / 2);

}

printEl(i - pattern, 'w', pattern, rows, colums);

printEl((i - pattern) + 1, 'r', pattern, rows, colums);

Sleep(speed / 2);

}

printEl(end - pattern, 'g', pattern, rows, colums);

--end;

if (!isChanged) {

break;

}

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

printEl(i - pattern, 'r', pattern, rows, colums);

printEl((i - pattern) - 1, 'y', pattern, rows, colums);

Sleep(speed / 2);

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

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

isChanged = true;

printEl(i - pattern, 'y', pattern, rows, colums);

printEl((i - pattern) - 1, 'r', pattern, rows, colums);

Sleep(speed / 2);

}

printEl(i - pattern, 'w', pattern, rows, colums);

printEl((i - pattern) - 1, 'r', pattern, rows, colums);

Sleep(speed / 2);

}

printEl(start - pattern, 'g', pattern, rows, colums);

++start;

}

SetConsoleTextAttribute(hStdout, FOREGROUND\_GREEN);

printMat(mat, rows, colums, 0, 0);

Sleep(speed);

SetConsoleTextAttribute(hStdout, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

}

void combSort(int\*\* mat, int\*\* pattern, int rows, int colums, int speed) {

system("cls");

printMat(mat, rows, colums, 0, 0);

int x, y, el;

float k = 1.247;

bool isChanged = true;

int\*\* end = &pattern[rows \* colums - 1], gap = rows\*colums - 1;

while (gap != 1) {

for (int\*\* i = pattern; i - pattern < (end - pattern) + 1 - gap; ++i) {

printEl(i - pattern, 'r', pattern, rows, colums);

printEl((i - pattern) + gap, 'b', pattern, rows, colums);

Sleep(speed/2);

if (\*\*i > \*\*(i + gap)) {

swap(\*\*(i + gap), \*\*i);

printEl(i - pattern, 'b', pattern, rows, colums);

printEl((i - pattern) + gap, 'r', pattern, rows, colums);

Sleep(speed / 2);

}

Sleep(speed/2);

printEl(i - pattern, 'w', pattern, rows, colums);

printEl((i - pattern) + gap, 'w', pattern, rows, colums);

}

gap /= k;

}

while (isChanged == true) {

isChanged = false;

for (int\*\* i = pattern; i < end; ++i) {

printEl(i - pattern, 'r', pattern, rows, colums);

printEl((i - pattern) + 1, 'y', pattern, rows, colums);

Sleep(speed / 2);

if (\*\*(i + 1) < \*\*i) {

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

isChanged = true;

printEl(i - pattern, 'y', pattern, rows, colums);

printEl((i - pattern) + 1, 'r', pattern, rows, colums);

Sleep(speed / 2);

}

printEl(i - pattern, 'w', pattern, rows, colums);

printEl((i - pattern) + 1, 'r', pattern, rows, colums);

Sleep(speed / 2);

}

--end;

}

SetConsoleTextAttribute(hStdout, FOREGROUND\_GREEN);

printMat(mat, rows, colums, 0, 0);

Sleep(speed);

SetConsoleTextAttribute(hStdout, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

}

void insertSort(int\*\* mat, int\*\* pattern, int rows, int colums, int speed) {

system("cls");

printMat(mat, rows, colums, 0, 0);

int x, y, el;

int\*\* end = &pattern[rows \* colums - 1], stepCount;

SetConsoleTextAttribute(hStdout, FOREGROUND\_GREEN);

for (int\*\* start = pattern + 1; start <= end; ++start) {

stepCount = 0;

while (\*(start - stepCount) != pattern[0] && \*\*(start - stepCount) < \*\*(start - 1 - stepCount)) {

swap(\*\*(start - stepCount), \*\*(start - 1 - stepCount));

printEl((start - pattern) - stepCount, 'g', pattern, rows, colums);

printEl((start - pattern) - 1 - stepCount, 'r', pattern, rows, colums);

Sleep(speed);

++stepCount;

}

printEl((start - pattern) - stepCount, 'g', pattern, rows, colums);

if (\*(start - stepCount) != pattern[0]) {

printEl((start - pattern) - 1 - stepCount, 'g', pattern, rows, colums);

}

Sleep(speed);

}

}

void quickSortSep(int\*\* pattern, int colums, int rows, int\*\* start, int\*\* end, int speed) {

int el, x, y;

int\*\* pivot = start;

++start;

for (int\*\* i = pattern; i < pivot; ++i) {

printEl(i - pattern, 'g', pattern, rows, colums);

}

if (start >= end) {

return;

}

printEl(pivot - pattern, 'b', pattern, rows, colums);

Sleep(speed/2);

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

printEl(i - pattern, 'r', pattern, rows, colums);

Sleep(speed/2);

if (\*\*i < \*\*pivot) {

swap(\*\*i, \*\*start);

printEl(start - pattern, 'y', pattern, rows, colums);

++start;

}

if (start - 1 != i && i != end) {

printEl(i - pattern, 'v', pattern, rows, colums);

}

else if (start - 1 != i && i - pattern == rows \* colums - 1) {

printEl(i - pattern, 'w', pattern, rows, colums);

}

else if (start - 1 != i) {

printEl(i - pattern, 'g', pattern, rows, colums);

}

Sleep(speed/2);

}

swap(\*\*pivot, \*\*(start - 1));

for (int\*\* i = pivot; i < end; ++i) {

printEl(i - pattern, 'w', pattern, rows, colums);

}

printEl((start - pattern) - 1, 'g', pattern, rows, colums);

quickSortSep(pattern, colums, rows, pivot, start - 1, speed);

quickSortSep(pattern, colums, rows, start, end, speed);

}

void quickSortMain(int\*\* mat, int\*\* pattern, int rows, int colums, int speed) {

system("cls");

printMat(mat, rows, colums, 0, 0);

quickSortSep(pattern, colums, rows, pattern, pattern + (rows \* colums - 1), speed);

}

void mergeSortStep(int\*\* pattern, int rows, int colums, int start, int end, int twoSortArr[], int speed) {

int parity = (end - start + 1) % 2;

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

printEl(i, 'y', pattern, rows, colums);

}

Sleep(speed\*2);

if (end - start <= 1) {

if (end - start == 1 && \*pattern[end] < \*pattern[start]) {

printEl(start, 'v', pattern, rows, colums);

printEl(end, 'r', pattern, rows, colums);

Sleep(speed);

swap(\*pattern[start], \*pattern[end]);

printEl(start, 'r', pattern, rows, colums);

printEl(end, 'v', pattern, rows, colums);

Sleep(speed);

printEl(start, 'w', pattern, rows, colums);

printEl(end, 'w', pattern, rows, colums);

}

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

printEl(i, 'w', pattern, rows, colums);

}

return;

}

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

printEl(i, 'w', pattern, rows, colums);

}

Sleep(speed);

mergeSortStep(pattern, rows, colums, start, (end + start) / 2 - parity, twoSortArr, speed);

mergeSortStep(pattern, rows, colums, (end + start) / 2 + 1 - parity, end, twoSortArr, speed);

int ind = 0;

int lArrIndex = start, rArrIndex = (end + start) / 2 + 1 - parity;

for (int i = start; i <= (end + start) / 2 - parity; ++i) {

printEl(i, 'v', pattern, rows, colums);

}

for (int i = (end + start) / 2 + 1 - parity; i <= end; ++i) {

printEl(i, 'r', pattern, rows, colums);

}

Sleep(speed \* 2);

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

if (lArrIndex <= (end + start) / 2 - parity && rArrIndex <= end) {

if (\*pattern[lArrIndex] > \*pattern[rArrIndex]) {

twoSortArr[ind] = \*pattern[rArrIndex];

++rArrIndex;

++ind;

}

else {

twoSortArr[ind] = \*pattern[lArrIndex];

++lArrIndex;

++ind;

}

}

else if (rArrIndex > end) {

twoSortArr[ind] = \*pattern[lArrIndex];

++lArrIndex;

++ind;

}

else {

twoSortArr[ind] = \*pattern[rArrIndex];

++rArrIndex;

++ind;

}

}

ind = 0;

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

\*pattern[i] = twoSortArr[ind];

++ind;

}

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

printEl(i, 'g', pattern, rows, colums);

}

Sleep(speed \* 3);

}

void mergeSortMain(int\*\* mat, int\*\* pattern, int rows, int colums, int speed) {

system("cls");

printMat(mat, rows, colums, 0, 0);

int\* twoSortArr = new int[rows \* colums];

mergeSortStep(pattern, rows, colums, 0, rows \* colums - 1, twoSortArr, speed);

delete[] twoSortArr;

}

void multiplicationMat(int\* firstArr,int\*\* mat, int rows, int &colums, int secondMatColums, int speed){

system("cls");

int\*\* secondMat = new int\* [colums];

int\* secondArr = new int[colums \* secondMatColums];

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

secondMat[i] = secondArr + i \* secondMatColums;

}

srand(time(0));

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

secondArr[i] = rand() % (colums \* secondMatColums) + 1;

}

printMat(mat, rows, colums, 0, 0);

destCoord.X = (colums + 1)\*5;

destCoord.Y = rows / 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << "\*" << '\r';

cout.flush();

printMat(secondMat, rows, secondMatColums, (colums + 2)\*5, 0);

destCoord.X = (colums + 3 + secondMatColums) \* 5;

destCoord.Y = rows / 2;

SetConsoleCursorPosition(hStdout, destCoord);

cout << "=" << '\r';

cout.flush();

int\*\* multipMat = new int\* [rows];

int\* multipArr = new int[rows \* secondMatColums];

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

multipMat[i] = multipArr + i \* secondMatColums;

}

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

multipArr[i] = 0;

}

int \*firP = &mat[0][0], \* secP = secondArr, \* mulP = multipArr;

int x = (colums + 4 + secondMatColums) \* 5, y = 0;

int c;

for (;mulP - multipArr < rows \* secondMatColums; ++mulP) {

for (int i = 0; i < colums; ++firP, secP += secondMatColums, ++i) {

\*mulP += (\*firP) \* (\*secP);

c = 1;

}

secP += 1 - secondMatColums \* colums;

firP -= colums;

printNumber(\*mulP, x, y);

if ((mulP - multipArr + 1) % secondMatColums == 0 ) {

firP += colums;

y++;

secP -= secondMatColums;

x -= secondMatColums \* 5;

}

x+= 5;

Sleep(speed\*2);

}

if (firstArr) {

delete[] firstArr;

}

delete[] secondArr;

int\* arr = new int[rows \* secondMatColums];

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

mat[i] = arr + i \* secondMatColums;

}

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

arr[i] = multipArr[i];

}

delete[] multipArr;

delete[] secondMat;

delete[] multipMat;

colums = secondMatColums;

cin >> rows;

}

int main(){

setlocale(0, "");

int rows, colums, secondMatColums, number;

int speed = 100;

int choice = 0;

cout << "Введите через пробел кол-во строк и стобцов:\n";

cin >> rows >> colums;

int\*\* mat = new int\*[rows];

int\* arr = new int [rows \* colums];

int\*\* pattern = new int\* [rows \* colums];

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

mat[i] = arr + i \* colums;

}

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

arr[i] = 0;

}

do{

system("cls");

cout << "Ваша матрица:";

printMat(mat, rows, colums, 0, 1);

cout << "\nДействия:\n"

<< "1) Заполнить\n"

<< "2) Переставить блоки\n"

<< "3) Увеличить/уменьшить все элементы в/на n раз/число\n"

<< "4) Отсортировать\n"

<< "5) Умножить на другую матрицу\n"

<< "6) Изменить скорость отрисовки\n"

<< "-1) Закончить программу\n";

cin >> choice;

system("cls");

switch (choice) {

case 1: // Заполнить

do {

cout << "Выберите способ заполнения матрицы:\n"

<< "1) По спирали\n"

<< "2) По вертикали\n"

<< "3) Слева направо\n"

<< "0) Назад\n";

cin >> choice;

switch (choice) {

case 1:

fillMatSpiral(mat, rows, colums, speed);

break;

case 2:

fillMatVert(mat, rows, colums, speed);

break;

case 3:

fillMatSimple(mat, rows, colums, speed);

break;

}

} while (choice != 0);

break;

case 2: // Переставить блоки

do {

cout << "Как переставить блоки?:\n"

<< "1) По кругу\n"

<< "2) Наискось\n"

<< "3) По вертикали\n"

<< "4) По горизонтали\n"

<< "0) Назад\n";

cin >> choice;

system("cls");

cout << "старая матрица";

printMat(mat, rows, colums, 0, 1);

switch (choice) {

case 1:

replaceBlocksRound(mat, rows, colums);

break;

case 2:

replaceBlocksObliq(mat, rows, colums);

break;

case 3:

replaceBlocksVert(mat, rows, colums);

break;

case 4:

replaceBlocksHor(mat, rows, colums);

break;

}

cout << "\nновая матрица";

printMat(mat, rows, colums, 0, rows + 2);

cout << endl;

} while (choice != 0);

break;

case 3: // Увеличить/уменьшить все элементы в/на n раз/число

cout << "Введите число:\n";

cin >> number;

do {

cout << "\nДействия:\n"

<< "1) Прибавить " << number << "\n"

<< "2) Отнять " << number << "\n"

<< "3) Умножить на " << number << "\n"

<< "4) Разделить на " << number << "\n"

<< "5) Изменить число\n"

<< "0) Назад\n";

cin >> choice;

if (choice != 5){

system("cls");

cout << "старая матрица";

printMat(mat, rows, colums, 0, 1);

switch (choice) {

case 0:

break;

case 1:

arithOperMat(mat, rows, colums, '+', number);

break;

case 2:

arithOperMat(mat, rows, colums, '-', number);

break;

case 3:

arithOperMat(mat, rows, colums, '\*', number);

break;

case 4:

arithOperMat(mat, rows, colums, '/', number);

break;

}

cout << "\nновая матрица";

printMat(mat, rows, colums, 0, rows + 2);

cout << endl;

}

else {

system("cls");

cout << "Введите число:\n";

cin >> number;

}

} while (choice != 0);

break;

case 4: // Отсортировать

do {

system("cls");

cout << "Ваша матрица:";

printMat(mat, rows, colums, 0, 1);

cout << "\nКак отсортировать?:\n"

<< "1) Спиралью\n"

<< "2) Вертикально(вниз-вверх-вниз-...)\n"

<< "3) Слева направо\n"

<< "4) Наискосок\n"

<< "0) Назад\n";

cin >> choice;

switch (choice) {

case 1:

createSpiralPointersPattern(pattern, mat, rows, colums);

break;

case 2:

createVertPointersPattern(pattern, mat, rows, colums);

break;

case 3:

createVertPointersSimple(pattern, mat, rows, colums);

break;

case 4:

createVertPointersObliq(pattern, mat, rows, colums);

break;

}

if ( 0 < choice < 5) {

break;

}

} while (choice != 0);

while (choice != 0){

system("cls");

cout << "Ваша матрица:";

printMat(mat, rows, colums, 0, 1);

cout << "\nКакую сортировку использовать?:\n"

<< "1) Bubble sort\n"

<< "2) Shaker sort\n"

<< "3) Comb sort\n"

<< "4) Insert sort\n"

<< "5) Quck sort\n"

<< "6) Merge sort\n"

<< "0) Назад\n";

cin >> choice;

switch (choice) {

case 1:

bubbleSort( mat, pattern, rows, colums, speed);

break;

case 2:

shakerSort(mat, pattern, rows, colums, speed);

break;

case 3:

combSort(mat, pattern, rows, colums, speed);

break;

case 4:

insertSort(mat, pattern, rows, colums, speed);

break;

case 5:

quickSortMain(mat, pattern, rows, colums, speed);

break;

case 6:

mergeSortMain(mat, pattern, rows, colums, speed);

break;

}

}

break;

case 5: // Умножить на другую матрицу

cout << "Введите кол-во стобцов матрицы на которую будете умножать : ";

cin >> secondMatColums;

multiplicationMat(arr, mat, rows, colums, secondMatColums, speed);

arr = &mat[0][0];

break;

case 6: // Изменить скорость отрисовки

cout << "нынешнее значение : " << speed << "\n";

cout << "Введите желаемое значение(0 - очень быстро 300< - очень медленно ) : ";

cin >> speed;

break;

}

} while (choice != -1);

delete[] arr;

delete[] mat;

return 0;

}