**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. 3372 |  | Матвиенко А. В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Научиться пользоваться функциями, двумерными массивами, ссылками и указателями. Научиться заполнять матрицу через указатели, разделять ее на блоки и переставлять их, а также проводить различные операции с элементами матрицы. Научиться выполнять сортировки, используя арифметику указателей.

**Основные теоретические положения.**

### Определение функции

Функция определяет действия, которые выполняет программа. Функции позволяют выделить набор инструкций и придать ему имя. А затем многократно по присвоенному имени вызывать в различных частях программы. По сути функция - это именованный блок кода.

Формальное определение функции выглядит следующим образом:

тип имя\_функции(параметры)

{

инструкции

}

Первая строка представляет заголовок функции. Вначале указывается возвращаемый тип функции. Если функция не возвращает никакого значения, то используется тип void.

Затем идет имя функции, которое представляет произвольный идентификатор. К именованию функции применяются те же правила, что и к именованию переменных.

После имени функции в скобках идет перечисление параметров. Функция может не иметь параметров, в этом случае указываются пустые скобки.

После заголовка функции в фигурных скобках идет тело функции, которое содержит выполняемые инструкции.

Для возвращения результата функция применяет оператор return. Если функция имеет в качестве возвращаемого типа любой тип, кроме void, то она должна обязательно с помощью оператора return возвращать какое-либо значение.

### Выполнение функции

Для выполнения функции ее необходимо вызвать. Вызов функции осуществляется в форме:

имя\_функции(аргументы);

### Функции с переменным числом параметров

Одним из представителей подобных функций является, например, функция **printf ()**, используемая в языках C и C++ для вывода данных:

void main()

{

int a = 10, b = 20;

char \*str = "Это текст";

printf("a = %d, с = %s, b = %d", a, str, b);

}

Эта функция может принимать произвольное число аргументов в зависимости от структуры первого аргумента, задающего так называемую строку формата. В строке  формата каждый символ **%** означает подстановку соответствующего аргумента из перечисленных за строкой формата. Количество подставляемых аргументов должно соответствовать количеству символов **%** в стоке формата и может быть любым.

### Передача данных по значению

Механизм передачи данных через параметры функции очень прост. При вызове функции в определенной области памяти (в стеке программы) для каждого параметра функции создается переменная соответствующая типу данных параметра. В эти переменные копируются значения аргументов, использовавшихся при вызове функции. При выполнении кода функции эти копии значений аргументов могут использоваться для обработки, могут изменять свои значения, но эти изменения никак не затрагивают значений самих аргументов. Поэтому после завершения работы функции, значения аргументов, которые были использованы при вызове функции, останутся такими же, какими они были до вызова функции. Например:

void F (int I)

{

    I = I + 20;

    cout << I << endl;

}

int main()

{

    int A = 10;

    cout << A << endl;  //  Выведено значение 10

    F ( A );                      //  Выведено значение 30

    cout << A << endl;  //  Выведено значение 10

    return 0;

}

### Рекурсивное использование функций

Функции внутри своего тела могут вызывать сами себя. Такой вызов называется **рекурсией**. На основе рекурсии можно строить очень интересные алгоритмы обработки данных.

Рассмотрим функцию возведения вещественного значения **D** в целую положительную степень **P**. Очевидная реализация этой функции основана на использовании цикла:

double  Pow (double  D, unsigned P)

{

     double R = 1;

     for (unsigned I = 0; I < P; ++ I)

           R \*= D;

     return R;

}

А вот та же самая функция, реализованная на основе рекурсии:

double  Pow (double  D, unsigned P)

{

     if  (P)

          return  Pow ( D, P – 1 ) \* D;

     else

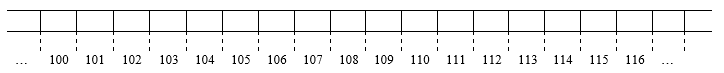
          return  1;

}

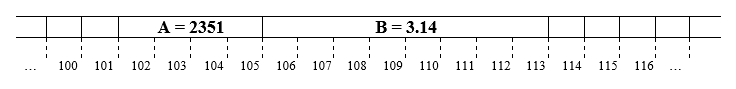
**Понятие указателей**

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.

Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения.

### Присваивание

Указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя.

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

int a = 1917;

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

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**

cout << Arr:

мы увидим некоторое целое значение в шестнадцатеричном формате, соответствующее адресу первого элемента этого массива.

**Постановка задачи.**

Используя арифметику указателей, заполнить 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):



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



Используя арифметику указателей, отсортировать элементы любой сортировкой из списка  
Варианты сортировок:

* Shaker sort;
* Comb sort;
* Insert sort;
* Quick sort;

Уменьшить, увеличить, умножить или поделить все элементы матрицы на введенное пользователем число (\*Для манипуляции с элементами используйте только арифметику указателей).

**Выполнение работы.**

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| Меню | |
| При запуске программы перед пользователем появляется окно с меню, где он может выбрать номер задания последовательности. | Меню:    Проверка на ввод символов, которые не входят в диапазон выбора: |

Продолжение Таблицы

|  |  |
| --- | --- |
| Выполнение 1 задания | |
| При вводе пользователем корректного значения пункта меню программа выполняется. | Заполнение выбранным пользователем способом матрицы. Порядок матрицы также определяется пользователем. |

|  |  |
| --- | --- |
| Выполнение 2 задания | |
| Пользователь выбирает порядок матрицы и способ переставления блоков | Разделение матрицы на блоки и их переставление 4 способами |

|  |  |
| --- | --- |
| Выполнение 3 задания | |
| Производится сортировка матрицы. |  |
| Выполнение 4 задания | |
| Пользователем вводится число и действие с матрицей, после чего это действие выполняется с заданным числом. |  |

**Выводы.**

В процессе выполнения работы я научился пользоваться функциями, двумерными массивами, ссылками и указателями. Научился заполнять матрицу через указатели, разделять ее на блоки и переставлять их, а также проводить различные операции с элементами матрицы. Научился выполнять сортировки, используя арифметику указателей.

Приложение А

рабочий код

#include <iostream>

#include <time.h>

#include <iomanip>

#include <Windows.h>

#include <conio.h>

using namespace std;

void createNewMatrix(int\* firstElement, int N)

{

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

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

{

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

}

}

void fillZeroMatrix(int\* firstElement, int N)

{

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

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

\*i = 0;

}

}

void spiralFill(int\* firstElement, int N)

{

int\* arr = new int[N \* N];

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

fillZeroMatrix(firstElement, N);

for (int index = 0; index < N \* N; index++)

{

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

{

int\* pUpLeft = firstElement + i \* (N + 1);

int\* pUpRight = firstElement + i \* (N - 1) + (N - 1);

int\* pDownRight = firstElement + (N \* N - 1) - i \* (N + 1);

int\* pDownLeft = firstElement + (N \* N - N) - i \* (N - 1);

for (int\* j = pUpLeft; j < pUpRight; j++)

{

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

arr[index] = \*j;

system("cls");

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

{

cout << setw(4) << \*i;

if ((i - firstElement + 1) % N == 0)

{

cout << "\n";

}

}

index++;

Sleep(100);

}

for (int\* j = pUpRight; j < pDownRight; j += N)

{

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

arr[index] = \*j;

system("cls");

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

{

cout << setw(4) << \*i;

if ((i - firstElement + 1) % N == 0)

{

cout << "\n";

}

}

index++;

Sleep(100);

}

for (int\* j = pDownRight; j > pDownLeft; j--)

{

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

arr[index] = \*j;

system("cls");

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

{

cout << setw(4) << \*i;

if ((i - firstElement + 1) % N == 0)

{

cout << "\n";

}

}

index++;

Sleep(100);

}

for (int\* j = pDownLeft; j > pUpLeft; j -= N)

{

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

arr[index] = \*j;

system("cls");

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

{

cout << setw(4) << \*i;

if ((i - firstElement + 1) % N == 0)

{

cout << "\n";

}

}

index++;

Sleep(100);

}

}

}

cout << endl;

}

void snakeFill(int\* firstElement, int N)

{

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

fillZeroMatrix(firstElement, N);

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

{

int\* pUp = firstElement + i \* 2;

int\* pDown = firstElement + N \* N - N + i \* 2;

int\* pRightUp = firstElement + N \* N - N + 1 + i \* 2;

int\* pRightDown = firstElement + 1 + i \* 2;

for (int\* j = pUp; j <= pDown; j += N)

{

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

system("cls");

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

{

cout << setw(4) << \*i;

if ((i - firstElement + 1) % N == 0)

{

cout << "\n";

}

}

Sleep(100);

}

for (int\* j = pRightUp; j >= pRightDown; j -= N)

{

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

system("cls");

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

{

cout << setw(4) << \*i;

if ((i - firstElement + 1) % N == 0)

{

cout << "\n";

}

}

Sleep(100);

}

}

}

void splitMatrix(int\* firstElement, int N)

{

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

int\* start = firstElement + N / 2 - 1;

int\* mid = firstElement + (N \* N / 2) - 1;

cout << "\n\n";

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

{

cout << setw(4) << \*i;

if (((i - firstElement + 1) % (N / 2) == 0) && ((i - firstElement + 1) % N != 0))

{

cout << " ";

}

if (((i - firstElement + 1) % ((N \* N) / 2) == 0) && ((i - firstElement + 1) % (N \* N) != 0))

{

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

}

if ((i - firstElement + 1) % N == 0)

{

cout << '\n';

}

}

}

void swapBlocksA(int\* firstElement, int N)

{

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

{

int\* ptr1 = firstElement + (i / (N / 2)) \* N + (i % (N / 2));

int\* ptr2 = ptr1 + (N / 2);

int\* ptr4 = ptr2 + (N / 2) \* N;

int\* ptr3 = ptr4 - (N / 2);

int temp = \*ptr2;

\*ptr2 = \*ptr1;

int temp1 = \*ptr4;

\*ptr4 = temp;

temp = \*ptr3;

\*ptr3 = temp1;

\*ptr1 = temp;

}

splitMatrix(firstElement, N);

}

void swapBlocksB(int\* firstElement, int N)

{

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

{

int\* ptr1 = firstElement + (i / (N / 2)) \* N + (i % (N / 2));

int\* ptr2 = ptr1 + (N / 2);

int\* ptr4 = ptr2 + (N / 2) \* N;

int\* ptr3 = ptr4 - (N / 2);

swap(\*ptr1, \*ptr4);

swap(\*ptr2, \*ptr3);

}

splitMatrix(firstElement, N);

}

void swapBlocksC(int\* firstElement, int N)

{

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

{

int\* ptr1 = firstElement + (i / (N / 2)) \* N + (i % (N / 2));

int\* ptr2 = ptr1 + (N / 2);

int\* ptr4 = ptr2 + (N / 2) \* N;

int\* ptr3 = ptr4 - (N / 2);

swap(\*ptr1, \*ptr3);

swap(\*ptr2, \*ptr4);

}

splitMatrix(firstElement, N);

}

void swapBlocksD(int\* firstElement, int N)

{

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

{

int\* ptr1 = firstElement + (i / (N / 2)) \* N + (i % (N / 2));

int\* ptr2 = ptr1 + (N / 2);

int\* ptr4 = ptr2 + (N / 2) \* N;

int\* ptr3 = ptr4 - (N / 2);

swap(\*ptr1, \*ptr2);

swap(\*ptr3, \*ptr4);

}

splitMatrix(firstElement, N);

}

void bubbleSort(int\* firstElement, int N)

{

int\* end = firstElement + (N \* N);

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

{

for (int\* j = firstElement; j < end - 1; j++)

{

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

{

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

}

}

}

}

void showMatrix(int\* firstElement, int N)

{

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

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

cout << setw(4) << \*i;

if ((i - firstElement + 1) % N == 0) {

cout << '\n';

}

}

}

void plusNumber(int\* firstElement, int N, int number)

{

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

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

\*i += number;

}

}

void minusNumber(int\* firstElement, int N, int number)

{

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

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

\*i -= number;

}

}

void multiplyNumber(int\* firstElement, int N, int number)

{

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

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

\*i \*= number;

}

}

void divideNumber(int\* firstElement, int N, int number)

{

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

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

\*i /= number;

}

}

void showIDZMatrix(int\* firstElement, int N)

{

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

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

cout << setw(4) << \*i;

if ((i - firstElement + 1) % N == 0) {

cout << '\n';

}

}

}

int main()

{

setlocale(LC\_ALL, "RU");

const int size6 = 6, size8 = 8, size10 = 10;

int arr6[size6][size6], arr8[size8][size8], arr10[size10][size10];

int\* pArr6 = &arr6[0][0], \* pArr8 = &arr8[0][0], \* pArr10 = &arr10[0][0];

while (true)

{

cin.clear();

cin.sync();

cout << "1) Используя арифметику указателей, заполнить 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):" << endl;

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

cout << "3) Используя арифметику указателей, отсортировать 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). " << endl;

cout << "4) Уменьшить, увеличить, умножить или поделить все элементы матрицы на введенное пользователем число (\*Для манипуляции с элементами используйте только арифметику указателей)." << endl;

int taskNumb;

cin >> taskNumb;

switch (taskNumb)

{

case 1:

{

cout << "Выберите способ заполнения матрицы: " << endl;

cout << "1) Спиралью\n" << "2) Змейкой" << endl;

int choise;

cin >> choise;

if (choise == 1)

{

cout << "Введите порядок матрицы (6, 8, 10)" << endl;

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

spiralFill(pArr6, size6);

break;

}

case 8:

{

spiralFill(pArr8, size8);

break;

}

case 10:

{

spiralFill(pArr10, size10);

break;

}

default:

{

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

}

}

if (choise == 2)

{

cout << "Введите порядок матрицы (6, 8, 10)" << endl;

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

snakeFill(pArr6, size6);

break;

}

case 8:

{

snakeFill(pArr8, size8);

break;

}

case 10:

{

snakeFill(pArr10, size10);

break;

}

default:

{

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

}

}

}

break;

case 2:

{

cout << "Изначальное расположение блоков: \n"

<< "\t --- --- \n"

<< "\t| 1 | 2 |\n"

<< "\t --- --- \n"

<< "\t| 4 | 3 |\n"

<< "\t --- --- \n\n";

cout << "Выберите способ переставления блоков:\n";

cout << "1) \n"

<< "\t --- --- \n"

<< "\t| 4 | 1 |\n"

<< "\t --- --- \n"

<< "\t| 3 | 2 |\n"

<< "\t --- --- \n";

cout << "2) \n"

<< "\t --- --- \n"

<< "\t| 3 | 4 |\n"

<< "\t --- --- \n"

<< "\t| 2 | 1 |\n"

<< "\t --- --- \n";

cout << "3) \n"

<< "\t --- --- \n"

<< "\t| 4 | 3 |\n"

<< "\t --- --- \n"

<< "\t| 1 | 2 |\n"

<< "\t --- --- \n";

cout << "4) \n"

<< "\t --- --- \n"

<< "\t| 2 | 1 |\n"

<< "\t --- --- \n"

<< "\t| 3 | 4 |\n"

<< "\t --- --- \n";

int choise;

cin >> choise;

if (choise == 1)

{

cout << "Введите порядок матрицы (6, 8, 10)\n";

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

cout << "Исходная матрица:";

createNewMatrix(pArr6, size6);

splitMatrix(pArr6, size6);

cout << "\nМатрица с переставленными блоками:";

swapBlocksA(pArr6, size6);

break;

}

case 8:

{

cout << "Исходная матрица:";

createNewMatrix(pArr8, size8);

splitMatrix(pArr8, size8);

cout << "\nМатрица с переставленными блоками:";

swapBlocksA(pArr8, size8);

break;

}

case 10:

{

cout << "Исходная матрица:";

createNewMatrix(pArr10, size10);

splitMatrix(pArr10, size10);

cout << "\nМатрица с переставленными блоками:";

swapBlocksA(pArr10, size10);

break;

}

default:

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

}

if (choise == 2)

{

cout << "Введите порядок матрицы (6, 8, 10)\n";

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

cout << "Исходная матрица:";

createNewMatrix(pArr6, size6);

splitMatrix(pArr6, size6);

cout << "\nМатрица с переставленными блоками:";

swapBlocksB(pArr6, size6);

break;

}

case 8:

{

cout << "Исходная матрица:";

createNewMatrix(pArr8, size8);

splitMatrix(pArr8, size8);

cout << "\nМатрица с переставленными блоками:";

swapBlocksB(pArr8, size8);

break;

}

case 10:

{

cout << "Исходная матрица:";

createNewMatrix(pArr10, size10);

splitMatrix(pArr10, size10);

cout << "\nМатрица с переставленными блоками:";

swapBlocksB(pArr10, size10);

break;

}

default:

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

}

if (choise == 3)

{

cout << "Введите порядок матрицы (6, 8, 10)\n";

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

cout << "Исходная матрица:";

createNewMatrix(pArr6, size6);

splitMatrix(pArr6, size6);

cout << "\nМатрица с переставленными блоками:";

swapBlocksC(pArr6, size6);

break;

}

case 8:

{

cout << "Исходная матрица:";

createNewMatrix(pArr8, size8);

splitMatrix(pArr8, size8);

cout << "\nМатрица с переставленными блоками:";

swapBlocksC(pArr8, size8);

break;

}

case 10:

{

cout << "Исходная матрица:";

createNewMatrix(pArr10, size10);

splitMatrix(pArr10, size10);

cout << "\nМатрица с переставленными блоками:";

swapBlocksC(pArr10, size10);

break;

}

default:

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

}

if (choise == 4)

{

cout << "Введите порядок матрицы (6, 8, 10)\n";

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

cout << "Исходная матрица:";

createNewMatrix(pArr6, size6);

splitMatrix(pArr6, size6);

cout << "\nМатрица с переставленными блоками:";

swapBlocksD(pArr6, size6);

break;

}

case 8:

{

cout << "Исходная матрица:";

createNewMatrix(pArr8, size8);

splitMatrix(pArr8, size8);

cout << "\nМатрица с переставленными блоками:";

swapBlocksD(pArr8, size8);

break;

}

case 10:

{

cout << "Исходная матрица:";

createNewMatrix(pArr10, size10);

splitMatrix(pArr10, size10);

cout << "\nМатрица с переставленными блоками:";

swapBlocksD(pArr10, size10);

break;

}

default:

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

}

break;

}

case 3:

{

cout << "Введите порядок матрицы (6, 8, 10)\n";

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

createNewMatrix(pArr6, size6);

cout << "Исходная матрица:\n";

showMatrix(pArr6, size6);

bubbleSort(pArr6, size6);

cout << "Отсортированная матрица:\n";

showMatrix(pArr6, size6);

break;

}

case 8:

{

createNewMatrix(pArr8, size8);

cout << "Исходная матрица:\n";

showMatrix(pArr8, size8);

bubbleSort(pArr8, size8);

cout << "Отсортированная матрица:\n";

showMatrix(pArr8, size8);

break;

}

case 10:

{

createNewMatrix(pArr10, size10);

cout << "Исходная матрица:\n";

showMatrix(pArr10, size10);

bubbleSort(pArr10, size10);

cout << "Отсортированная матрица:\n";

showMatrix(pArr10, size10);

break;

}

default:

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

break;

}

case 4:

{

cout << "Введите число\n";

int number;

cin >> number;

cout << "Выберите действие:\n" << "1) +\n" << "2) -\n" << "3) \*\n" << "4) /\n";

int choice;

cin >> choice;

switch (choice)

{

case 1:

{

cout << "Введите порядок матрицы (6, 8, 10)\n";

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr6, size6);

showMatrix(pArr6, size6);

plusNumber(pArr6, size6, choice);

cout << "После манипуляции:\n";

showMatrix(pArr6, size6);

break;

}

case 8:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr8, size8);

showMatrix(pArr8, size8);

plusNumber(pArr8, size8, choice);

cout << "После манипуляции:\n";

showMatrix(pArr8, size8);

break;

}

case 10:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr10, size10);

showMatrix(pArr10, size10);

plusNumber(pArr10, size10, choice);

cout << "После манипуляции:\n";

showMatrix(pArr10, size10);

break;

}

default:

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

break;

}

case 2:

{

cout << "Введите порядок матрицы (6, 8, 10)\n";

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr6, size6);

showMatrix(pArr6, size6);

minusNumber(pArr6, size6, choice);

cout << "После манипуляции:\n";

showMatrix(pArr6, size6);

break;

}

case 8:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr8, size8);

showMatrix(pArr8, size8);

minusNumber(pArr8, size8, choice);

cout << "После манипуляции:\n";

showMatrix(pArr8, size8);

break;

}

case 10:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr10, size10);

showMatrix(pArr10, size10);

minusNumber(pArr10, size10, choice);

cout << "После манипуляции:\n";

showMatrix(pArr10, size10);

break;

}

default:

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

break;

}

case 3:

{

cout << "Введите порядок матрицы (6, 8, 10)\n";

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr6, size6);

showMatrix(pArr6, size6);

multiplyNumber(pArr6, size6, choice);

cout << "После манипуляции:\n";

showMatrix(pArr6, size6);

break;

}

case 8:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr8, size8);

showMatrix(pArr8, size8);

multiplyNumber(pArr8, size8, choice);

cout << "После манипуляции:\n";

showMatrix(pArr8, size8);

break;

}

case 10:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr10, size10);

showMatrix(pArr10, size10);

multiplyNumber(pArr10, size10, choice);

cout << "После манипуляции:\n";

showMatrix(pArr10, size10);

break;

}

default:

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

break;

}

case 4:

{

cout << "Введите порядок матрицы (6, 8, 10)\n";

int matrixOrder;

cin >> matrixOrder;

switch (matrixOrder)

{

case 6:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr6, size6);

showMatrix(pArr6, size6);

divideNumber(pArr6, size6, choice);

cout << "После манипуляции:\n";

showMatrix(pArr6, size6);

break;

}

case 8:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr8, size8);

showMatrix(pArr8, size8);

divideNumber(pArr8, size8, choice);

cout << "После манипуляции:\n";

showMatrix(pArr8, size8);

break;

}

case 10:

{

cout << "Исходная матрица:\n";

createNewMatrix(pArr10, size10);

showMatrix(pArr10, size10);

divideNumber(pArr10, size10, choice);

cout << "После манипуляции:\n";

showMatrix(pArr10, size10);

break;

}

default:

cout << "Доступны размерности 6, 8, 10.\n\n";

break;

}

break;

}

}

break;

}

default:

cout << "Доступно 4 задания.\n";

\_getch();

}

}

return 0;

}