МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра «Информационные системы»

**отчет**

по практической работе №3

по дисциплине «Программирование»

Тема: "Указатели"

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Голубцов В.В. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

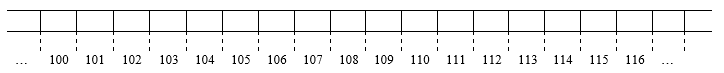
Цель работы.

Познакомиться на практике с указателями и арифметикой работы с ними.

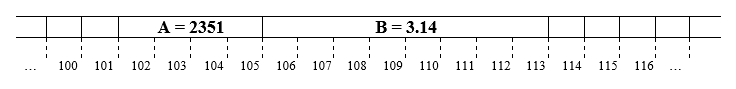
Основные теоретические положения.

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 101 и занимает 4 байта, а переменная **B** имеет адрес 105 и занимает 8 байт памяти.

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

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

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.

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.

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.

**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

int A1[10], A2[10];

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:

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. 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 +.

Постановка задачи.

Необходимо написать программу, которая:

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.



2)    Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами:



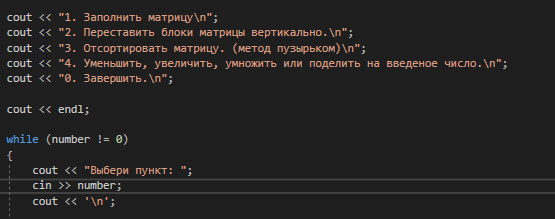
3)    Используя арифметику указателей, сортирует элементы любой сортировкой.

4)    Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

Выполнение работы.

Программа разделена на функции выполняющие операции над массивом данных.

Вначале пользователю необходимо выбрать операцию выполнения, всего их 4.



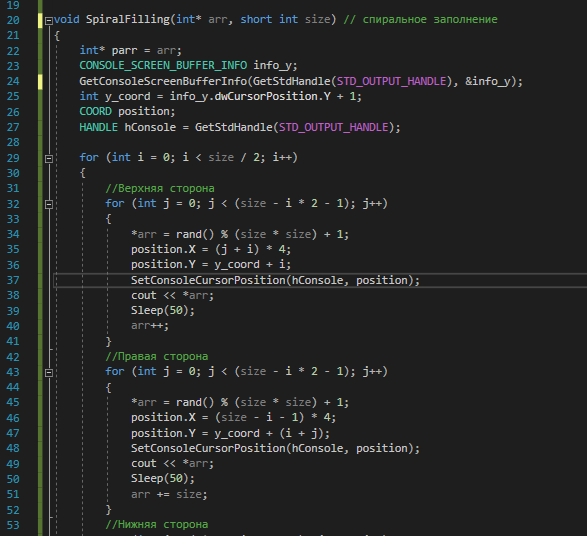
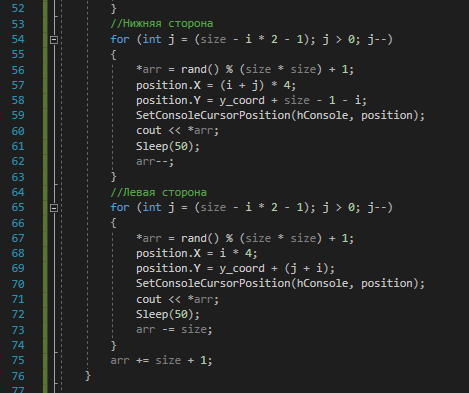
Первым пунктом мы заполняем массив элементов по спирали с поэтапным выводом их на экран с помощью задержки Sleep(50).

Рисунок : заполнение матрицы и вывод ее на экран



П

Рисунок : заполнение матрицы и вывод ее на экран

Принцип работы: от 0 до size/2 выполняются циклы. Вначале алгоритм проходит по верхней стороне, там указатель увеличивается на единицу, далее по правой, указатель увеличивается на значение, равное стороне матрицы, после по нижней стороне указатель уменьшается на единицу, и в конце - по левой стороне, где указатель снова уменьшается на значение, равное стороне матрицы. По завершению круга указатель смещается на единицу вниз и вперед.

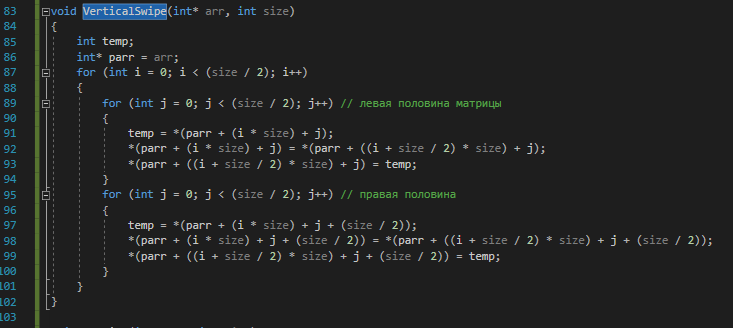
Следующий пункт это перестановка блоков матрицы. Я использовал перестановку по вертикали.

Рисунок : перестановка блоков

Здесь циклы разделены на правую и левую часть матрицы. Каждый выполняет следующие действия: запоминает значение элемента верхней половины и замещает его элементом из нижней половины, позиция которого соответствует ему в нижнем блоке.

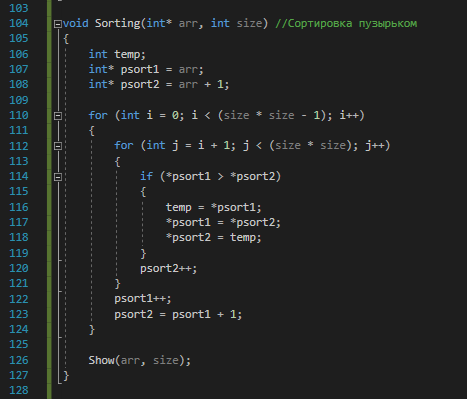
Далее идет сортировка массива элементов. Я выбрал сортировку пузырьком.

Рисунок : сортировка массива

Принцип работы это сравнение текущего элемента с следующим и если текущий меньше, то заменяем его на следующий. После завершения сортировки снова выводим матрицу на экран.

Финальный пункт это операции над каждым из элементов матрицы. Всего есть 4 типа операции, это: сложение, вычитание, умножение, деление.

Реализованы они перебором каждого элемента матрицы и выполнение действия с ним с помощью операций (+=, -=, \*=, /=). В конце снова выводим массив на экран.

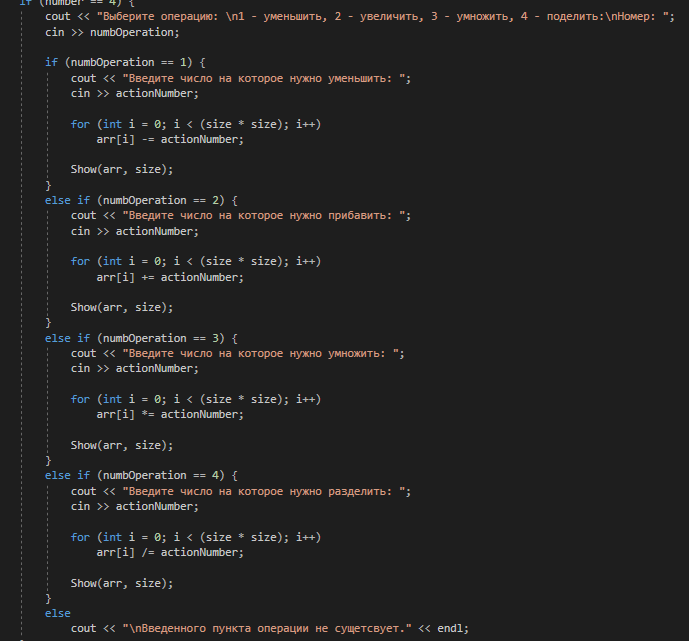


Рисунок : операции над элементами матрицы

Выводы.

Была написана программа реализующая способы работы с массивами посредством арифметики указателей, что дало понять и закрепить на практике их удобство работы с данными.

Полный код программы}

#include <iostream>

#include <stdio.h>

#include <Windows.h>

#include <time.h>

#include <iostream>

using namespace std;

void Show(int\* arr, int size)

{

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

{

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

cout << arr[(i \* size) + j] << ' ';

cout << '\n';

}

}

void SpiralFilling(int\* arr, short int size)

{

int\* parr = arr;

CONSOLE\_SCREEN\_BUFFER\_INFO info\_y;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &info\_y);

int y\_coord = info\_y.dwCursorPosition.Y + 1;

COORD position;

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

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

{

//Верхняя сторона

for (int j = 0; j < (size - i \* 2 - 1); j++)

{

\*arr = rand() % (size \* size) + 1;

position.X = (j + i) \* 4;

position.Y = y\_coord + i;

SetConsoleCursorPosition(hConsole, position);

cout << \*arr;

Sleep(50);

arr++;

}

//Правая сторона

for (int j = 0; j < (size - i \* 2 - 1); j++)

{

\*arr = rand() % (size \* size) + 1;

position.X = (size - i - 1) \* 4;

position.Y = y\_coord + (i + j);

SetConsoleCursorPosition(hConsole, position);

cout << \*arr;

Sleep(50);

arr += size;

}

//Нижняя сторона

for (int j = (size - i \* 2 - 1); j > 0; j--)

{

\*arr = rand() % (size \* size) + 1;

position.X = (i + j) \* 4;

position.Y = y\_coord + size - 1 - i;

SetConsoleCursorPosition(hConsole, position);

cout << \*arr;

Sleep(50);

arr--;

}

//Левая сторона

for (int j = (size - i \* 2 - 1); j > 0; j--)

{

\*arr = rand() % (size \* size) + 1;

position.X = i \* 4;

position.Y = y\_coord + (j + i);

SetConsoleCursorPosition(hConsole, position);

cout << \*arr;

Sleep(50);

arr -= size;

}

arr += size + 1;

}

position.X = 0;

position.Y = y\_coord + size;

SetConsoleCursorPosition(hConsole, position);

}

void VerticalSwipe(int\* arr, int size)

{

int temp;

int\* parr = arr;

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

{

for (int j = 0; j < (size / 2); j++) // левая половина матрицы

{

temp = \*(parr + (i \* size) + j);

\*(parr + (i \* size) + j) = \*(parr + ((i + size / 2) \* size) + j);

\*(parr + ((i + size / 2) \* size) + j) = temp;

}

for (int j = 0; j < (size / 2); j++) // правая половина

{

temp = \*(parr + (i \* size) + j + (size / 2));

\*(parr + (i \* size) + j + (size / 2)) = \*(parr + ((i + size / 2) \* size) + j + (size / 2));

\*(parr + ((i + size / 2) \* size) + j + (size / 2)) = temp;

}

}

}

void Sorting(int\* arr, int size) //Сортировка пузырьком

{

int temp;

int\* psort1 = arr;

int\* psort2 = arr + 1;

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

{

for (int j = i + 1; j < (size \* size); j++)

{

if (\*psort1 > \*psort2)

{

temp = \*psort1;

\*psort1 = \*psort2;

\*psort2 = temp;

}

psort2++;

}

psort1++;

psort2 = psort1 + 1;

}

Show(arr, size);

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

int number = 5;

int arr[100];

int size, numbOperation, actionNumber;

cout << "1. Заполнить матрицу\n";

cout << "2. Переставить блоки матрицы вертикально.\n";

cout << "3. Отсортировать матрицу. (метод пузырьком)\n";

cout << "4. Уменьшить, увеличить, умножить или поделить на введеное число.\n";

cout << "0. Завершить.\n";

cout << endl;

while (number != 0)

{

cout << "Выбери пункт: ";

cin >> number;

cout << '\n';

if (number == 1) { // создание матрицы

cout << "Выберите число: 6, 8, 10: ";

cin >> size;

if ((size == 6) || (size == 8) || (size == 10))

SpiralFilling(arr, size);

else

cout << "Вы ввели некорректное число\n";

}

if (number == 2) { // перемещение по вертикали

VerticalSwipe(arr, size);

Show(arr, size);

}

if (number == 3) { // сортировка

Sorting(arr, size);

}

if (number == 4) {

cout << "Выберите операцию: \n1 - уменьшить, 2 - увеличить, 3 - умножить, 4 - поделить:\nНомер: ";

cin >> numbOperation;

if (numbOperation == 1) {

cout << "Введите число на которое нужно уменьшить: ";

cin >> actionNumber;

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

arr[i] -= actionNumber;

Show(arr, size);

}

else if (numbOperation == 2) {

cout << "Введите число на которое нужно прибавить: ";

cin >> actionNumber;

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

arr[i] += actionNumber;

Show(arr, size);

}

else if (numbOperation == 3) {

cout << "Введите число на которое нужно умножить: ";

cin >> actionNumber;

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

arr[i] \*= actionNumber;

Show(arr, size);

}

else if (numbOperation == 4) {

cout << "Введите число на которое нужно разделить: ";

cin >> actionNumber;

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

arr[i] /= actionNumber;

Show(arr, size);

}

else

cout << "\nВведенного пункта операции не сущетсвует." << endl;

}

}

cout << "..." << endl;

}

Приложение Б

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

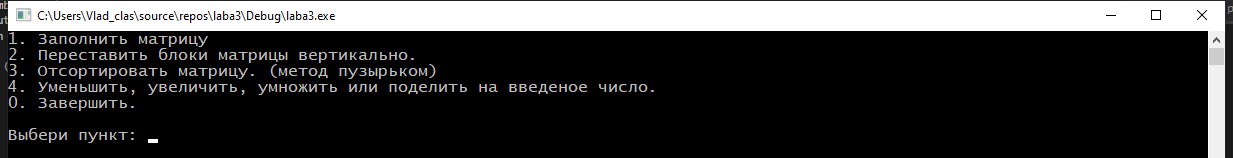
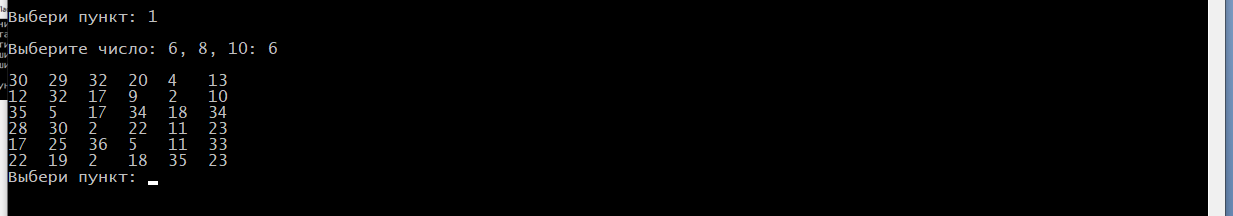
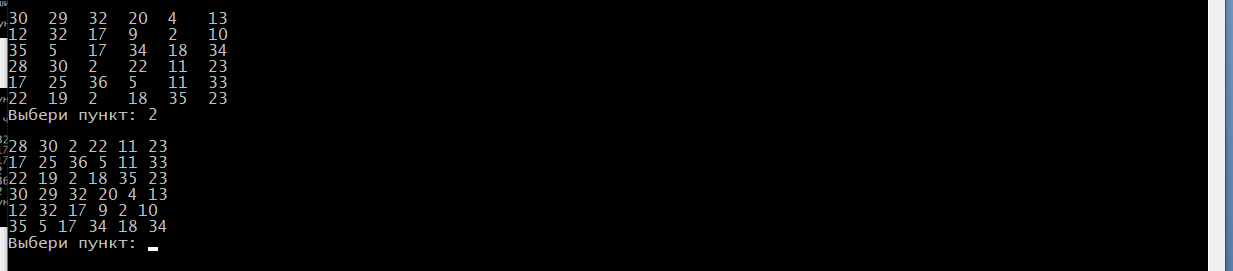
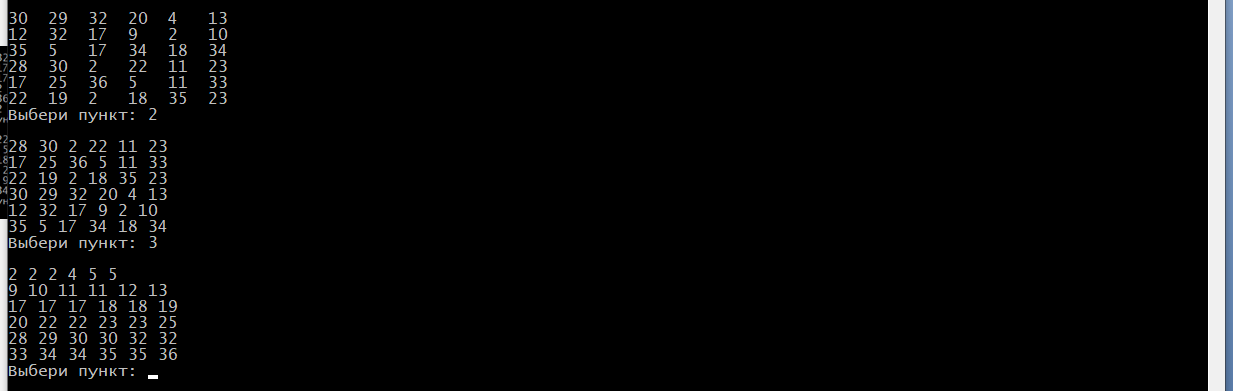


Рисунок 9:сортировка элементов матрицы

Рисунок 8:перестановка блоков по вертикали

Рисунок 7:отображение матрицы

Рисунок 6:меню выбора операции

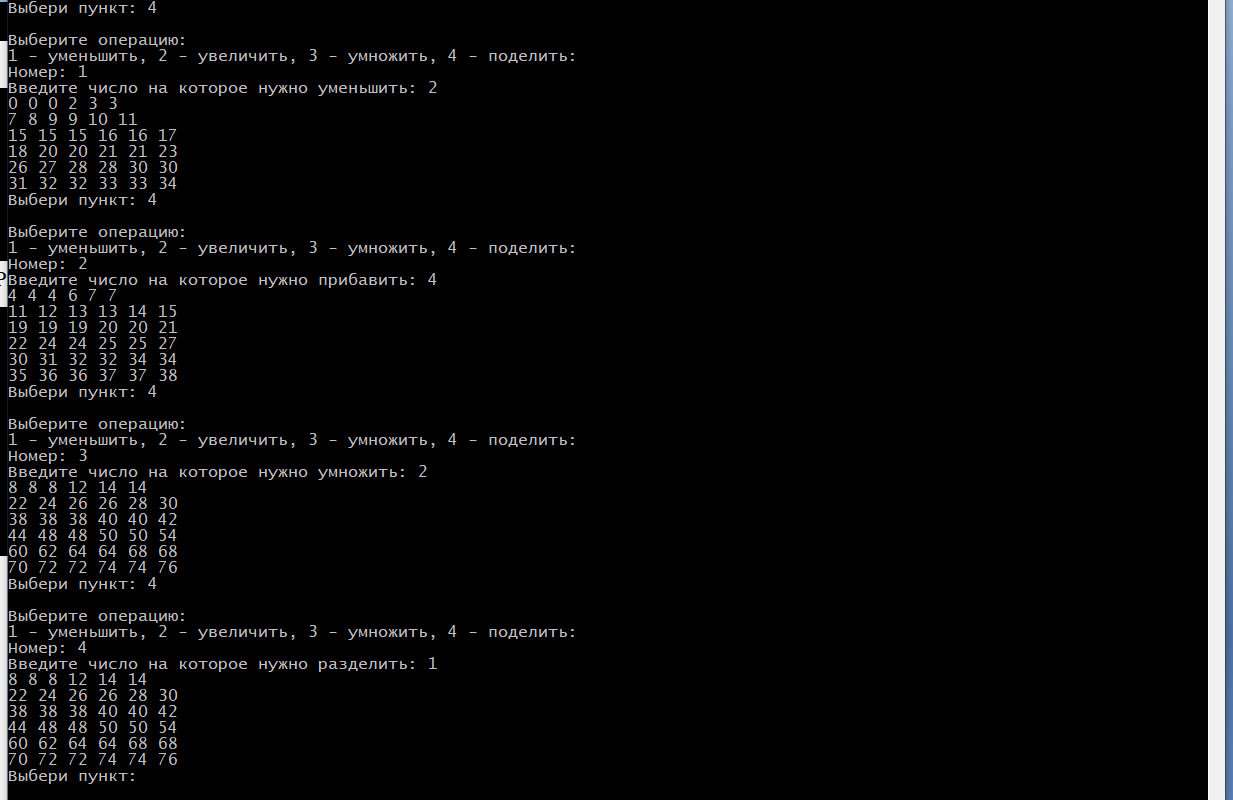


Рисунок 10: операции над элементами матрицы