МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Лабораторная работа №4 Рекурсия и головоломки

(наименование работы)

ОТЧЕТ

по лабораторной работе

по дисциплине

Технологии программирования

(наименование дисциплины)

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Капранов С.Н.

(подпись) (фамилия, и., о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сангалова О.С.

(подпись) (фамилия, и., о.)

18-ИСТ-4

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород, 2020

Оглавление

[Задача 3](#_Toc36938385)

[Листинг программы 3](#_Toc36938386)

[Результат работы программы 7](#_Toc36938387)

# Задача

Вариант 18 – Квадраты

## Листинг программы

**Файлы заголовков**

**board.h**

#ifndef BOARD\_H

#define BOARD\_H

#include "cell.h"

class **Board**

{

public:

**Board**();

~**Board**();

void **PrintBoard**();

void **Clear**();

bool **Solution**();

void **Answer**();

void **Input**();

private:

int\* mData;

int count\_of\_val;

int\* pos\_of\_val\_X;

int\* pos\_of\_val\_Y;

int mSize\_of\_board;

Cell\*\* mBoard;

};

#endif // BOARD\_H

**cell.h**

#ifndef CELL\_H

#define CELL\_H

class **Cell**

{

public:

**Cell**();

//Метод получения значения клетки

int **GetVal**() { return mVal; }

//Метод установки значения клетки

void **SetVal**(int val) { mVal = val; }

//Метод получения номера блока

int **GetNumOfVal**() { return num\_of\_mVal; }

//Метод установки номера блока

void **SetNumOfVal**(int val) { num\_of\_mVal = val; }

private:

int mVal;

int num\_of\_mVal;

};

#endif // CELL\_H

**get.h**

#ifndef GET\_H

#define GET\_H

class **Get**

{

private:

int intVal;

double dblVal;

public:

**Get**();

void **GetInt**(int\*);

void **GetInt**(int\*, int, int);

void **GetIntMax**(int\*, int);

void **GetIntMin**(int\*, int);

void **GetDbl**(double\*);

void **GetDbl**(double\*, double, double);

void **GetDblMax**(double\*, double);

void **GetDblMin**(double\*, double);

char\* **toString**(int \_number);

};

#endif // GET\_H

**board.cpp**

#include <iostream>

#include "board.h"

#include "get.h"

//Конструктор, принимает размер поля

//Необходим для выделения памяти и инициализаций

Board::**Board**()

{

//Инициализируем размер поля

mSize\_of\_board = 3;

//Выделяем память под массив значений

mData = new int[mSize\_of\_board \* 2];

//Присваеваем первое значение счетчика

count\_of\_val = -1;

//Выделяем память под координаты (С избытком, что бы избежать выхода за массив)

pos\_of\_val\_X = new int[mSize\_of\_board \* 2];

pos\_of\_val\_Y = new int[mSize\_of\_board \* 2];

//Собираем поле и выделяем под него память

mBoard = new Cell\* [mSize\_of\_board];

{

for(int i(0); i < mSize\_of\_board; i++)

{

mBoard[i] = new Cell [mSize\_of\_board];

}

}

//Инициализируем поле

for (int i(0); i < mSize\_of\_board; i++)

{

for(int j(0); j < mSize\_of\_board; j++)

{

mBoard[i][j].SetVal(0);

mBoard[i][j].SetNumOfVal(-1);

}

}

}

//Диструктор, служит для очистки памяти

Board::~**Board**()

{

for(int i(0); i < mSize\_of\_board; i++)

{

delete mBoard[i];

}

delete mBoard;

delete pos\_of\_val\_X;

delete pos\_of\_val\_Y;

delete mData;

}

//Метод печати поля

//Выводит поле в консоль

void Board::**PrintBoard**()

{

std::cout << "|\\";

for (int i = 0; i < mSize\_of\_board; i++)

{

if(i < 9)

{

std::cout << "|- ";

}

else

{

std::cout << "|-";

}

std::cout <<i + 1<< " -|";

}

std::cout << " " << std::endl;

for (int i = 0; i < mSize\_of\_board; i++)

{

if(i < 9)

{

std::cout << i + 1;

std::cout << " ";

}

else

{

std::cout << i + 1;

}

for (int j = 0; j < mSize\_of\_board; j++)

{

if(mBoard[i][j].GetVal() < 9)

{

std::cout << "|- ";

}

else

{

std::cout << "|-";

}

std::cout << mBoard[i][j].GetVal() << " -|";

}

std::cout << " " << std::endl;

}

std::cout << std::endl << std::endl;

}

//Метод очистки

//Выполняет откат в случае ошибки расположения фигуры в Метода Solution

void Board::**Clear**()

{

// Если была ошибка в подборе расположения то очищаем

for (int i(0); i < mSize\_of\_board; i++) // поле от последней фигуры и спускаем флаг.

{ // (Пересечение с другой фигурой)

for(int j(0); j < mSize\_of\_board; j++) //------------------------------------------

{

if(mBoard[i][j].GetVal() == 0 && mBoard[i][j].GetNumOfVal() == count\_of\_val)

{

mBoard[i][j].SetNumOfVal(-1);

}

}

}

}

//Метод решения

// рекурсивно перебирает все возможные варианты решения головоломки

bool Board::**Solution**()

{

//Флаг для контроля ошибок

int flag\_of\_error = 0;

for (int length(1); length < mSize\_of\_board + 1; length++) //------------------------------------------

{ // Подбор длины и ширины для фигуры

for (int width(1); width < mSize\_of\_board + 1; width++) // заданной площади

{ //

if(length \* width == mBoard[pos\_of\_val\_X[count\_of\_val]][pos\_of\_val\_Y[count\_of\_val]].GetVal()) //------------------------------------------

{

for(int prm\_l(1); prm\_l < length + 1; prm\_l++) //------------------------------------------

{ // Подбор местоположения фигуры

for(int prm\_w(1); prm\_w < width + 1; prm\_w++) //------------------------------------------

{

//Подбираем "параметры" для вычисления клеток принадлежащих фигуре

//Что в последствии покажет ее местоположение

int temp\_l = length;

int temp\_w = width;

temp\_l -= prm\_l;

temp\_w -= prm\_w;

for(int k(0); k < length; k++)

{

temp\_w = width - prm\_w;

for(int p(0); p < width; p++)

{

//Вычесляем каждую координату для нахождения клетки фигуры

int temp\_x = pos\_of\_val\_X[count\_of\_val] - temp\_l;

int temp\_y = pos\_of\_val\_Y[count\_of\_val] - temp\_w;

//Проверка выхода за пределы массива

if(temp\_x > -1 && temp\_y > -1 && (temp\_x >= mSize\_of\_board || temp\_y >= mSize\_of\_board))

{

flag\_of\_error = 1;

}

//Если клетка не пренадлежит другой фигуре, то отправляем значение блока в клетку

else if(temp\_x > -1 && temp\_y > -1 && (mBoard[temp\_x][temp\_y].GetNumOfVal() == -1 || mBoard[temp\_x][temp\_y].GetVal() == mData[count\_of\_val]))

{

mBoard[temp\_x][temp\_y].SetNumOfVal(count\_of\_val);

}

//В любом другом случае - ошибка

else

{

flag\_of\_error = 1;

}

temp\_w--;

}

temp\_l--;

}

//Если фигура расположилась на поле без ошибок

if(flag\_of\_error == 0)

{

//Если ещё есть фигуры

if(count\_of\_val > 0)

{

//Находим их местоположение

count\_of\_val--;

if(Solution())

{

//Если нашли, возвращаем true

count\_of\_val++;

return true;

}

else

{

//Если не нашли, подбираем другое положение для нашей фигуры

flag\_of\_error = 1;

}

}

else

{

//Если фигура последняя, возвращаем true

count\_of\_val++;

return true;

}

}

//Откат в случае ошибок

if(flag\_of\_error == 1)

{

Clear();

flag\_of\_error = 0;

}

}

}

}

}

}

//Если решение не было найдено

count\_of\_val++;

return false;

}

//Метод печати ответа на головоломку

//Считывает принадлежность к блоку, и ставит соответствующее число в клетку

void Board::**Answer**()

{

if(Solution())

{

for (int i(0); i < mSize\_of\_board; i++)

{

for(int j(0); j < mSize\_of\_board; j++)

{

mBoard[i][j].SetVal(mData[mBoard[i][j].GetNumOfVal()]);

}

}

}

}

//Метод ввода данных

//Принимает задачу от пользователя

void Board::**Input**()

{

//Временная переменная для передачи значения клетки

int temp\_val = 0;

//До тех пор пока пользователь не отменил ввод

while(temp\_val == 0)

{

//Счетчик кол-ва фигур ++

count\_of\_val++;

//Создаем объект для коректного ввода

Get get;

//Создаем временные переменные для координат

int temp\_X,temp\_Y;

//Принимаем данные

std::cout << "Input value of cell: ";

get.GetIntMin(&temp\_val, 1);

system("cls");

std::cout << "Input value of raw: ";

get.GetInt(&temp\_X, 1, mSize\_of\_board);

system("cls");

temp\_X--;

std::cout << "Input value of colomn: ";

get.GetInt(&temp\_Y, 1, mSize\_of\_board);

system("cls");

temp\_Y--;

//Отправляем данные по назначению

for (int i(0); i < mSize\_of\_board; i++)

{

for(int j(0); j < mSize\_of\_board; j++)

{

if(temp\_X == i && temp\_Y == j)

{

if(mBoard[i][j].GetVal() == 0)

{

//Забиваем позицию элемента в память класса

pos\_of\_val\_X[count\_of\_val] = i;

pos\_of\_val\_Y[count\_of\_val] = j;

//А так же значение

mData[count\_of\_val] = temp\_val;

//Проставляем номер блока клетки

mBoard[i][j].SetNumOfVal(count\_of\_val);

}

//Проставляем значение клетки

mBoard[i][j].SetVal(temp\_val);

}

}

}

//Печатаем поле

PrintBoard();

//Спрашиваем есть ли ещё данные

std::cout << "New value accept!\n For new value input \"0\"" << std::endl;

get.GetInt(&temp\_val);

}

}

**get.cpp**

#include "get.h"

#include <iostream>

using namespace std;

Get::**Get**()

{

intVal = 0;

dblVal = 0;

}

void Get::**GetInt**(int \*tmpVal)

{

cin >> \*tmpVal;

//Если не верно

if (!(cin))

{

//Подготавливаем поток к повторному вводу

cin.clear();

cin.sync();

cin.ignore();

\*tmpVal = - 1;

}

system("cls");

}

void Get::**GetInt**(int \*tmpVal, int min, int max)

{

//Если мин больше меняем местами

if(min > max)

{

max+=min;

min = max - min;

max = max - min;

}

//Вводи значение пока не будет верно

do{

//Вводим значение

cin >> \*tmpVal;

//Если не верно

if (!(cin))

{

//Подготавливаем поток к повторному вводу

cin.clear();

cin.sync();

cin.ignore();

\*tmpVal = min - 1;

}

system("cls");

cout << "Incorect, please repeat input: ";

} while (\*tmpVal < min || \*tmpVal > max);

system("cls");

system("cls");

}

void Get::**GetIntMax**(int \*tmpVal, int max)

{

//Вводи значение пока не будет верно

do{

//Вводим значение

cin >> \*tmpVal;

//Если не верно

if (!(cin))

{

//Подготавливаем поток к повторному вводу

cin.clear();

cin.sync();

cin.ignore();

\*tmpVal = - 1;

}

system("cls");

cout << "Incorect, please repeat input: ";

} while (\*tmpVal > max);

system("cls");

}

void Get::**GetIntMin**(int \*tmpVal, int min)

{

//Вводи значение пока не будет верно

do{

//Вводим значение

cin >> \*tmpVal;

//Если не верно

if (!(cin))

{

//Подготавливаем поток к повторному вводу

cin.clear();

cin.sync();

cin.ignore();

\*tmpVal = - 1;

}

system("cls");

cout << "Incorect, please repeat input: ";

} while (\*tmpVal < min);

system("cls");

}

void Get::**GetDbl**(double \*tmpVal)

{

cin >> \*tmpVal;

//Если не верно

if (!(cin))

{

//Подготавливаем поток к повторному вводу

cin.clear();

cin.sync();

cin.ignore();

\*tmpVal = - 1.0;

}

system("cls");

}

void Get::**GetDbl**(double \*tmpVal, double min, double max)

{

//Если мин больше меняем местами

if(min > max)

{

max+=min;

min = max - min;

max = max - min;

}

//Вводи значение пока не будет верно

do{

//Вводим значение

cin >> \*tmpVal;

//Если не верно

if (!(cin))

{

//Подготавливаем поток к повторному вводу

cin.clear();

cin.sync();

cin.ignore();

\*tmpVal = min - 1;

}

system("cls");

cout << "Incorect, please repeat input: ";

} while (\*tmpVal < min || \*tmpVal > max);

system("cls");

}

void Get::**GetDblMax**(double \*tmpVal, double max)

{

//Вводи значение пока не будет верно

do{

//Вводим значение

cin >> \*tmpVal;

//Если не верно

if (!(cin))

{

//Подготавливаем поток к повторному вводу

cin.clear();

cin.sync();

cin.ignore();

\*tmpVal = - 1;

}

system("cls");

cout << "Incorect, please repeat input: ";

} while (\*tmpVal > max);

system("cls");

}

void Get::**GetDblMin**(double \*tmpVal, double min)

{

//Вводи значение пока не будет верно

do{

//Вводим значение

cin >> \*tmpVal;

//Если не верно

if (!(cin))

{

//Подготавливаем поток к повторному вводу

cin.clear();

cin.sync();

cin.ignore();

\*tmpVal = - 1;

}

system("cls");

cout << "Incorect, please repeat input: ";

} while (\*tmpVal < min);

system("cls");

}

char\* Get::**toString**(int \_number) {

int m = \_number;

int k = 0;

char \*result;

while(m!=0) {

k++;

m = m/10;

}

result = new char[k+1];

sprintf(result, "%d", \_number);

return result;

}

**main.cpp**

#include <iostream>

#include "board.h"

int **main**()

{

Board board;

board.PrintBoard();

board.Input();

board.PrintBoard();

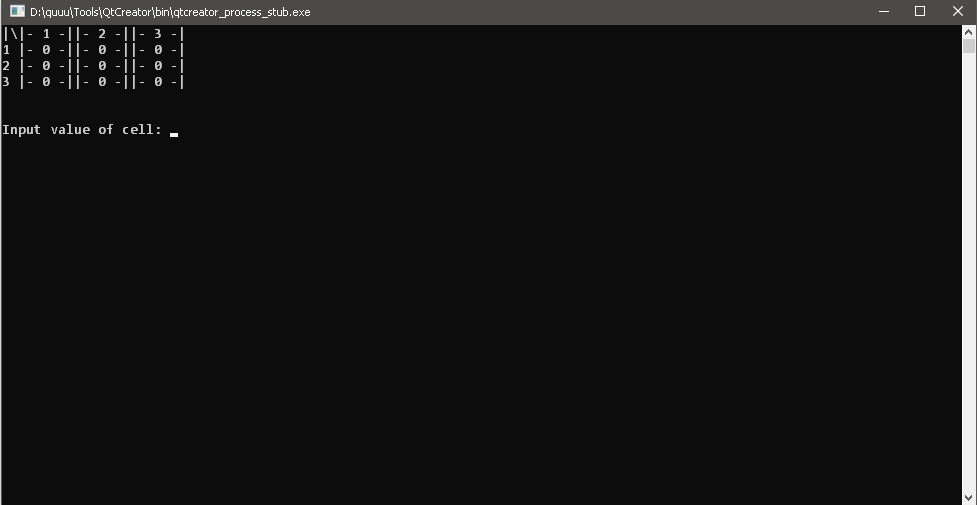
board.Answer();

board.PrintBoard();

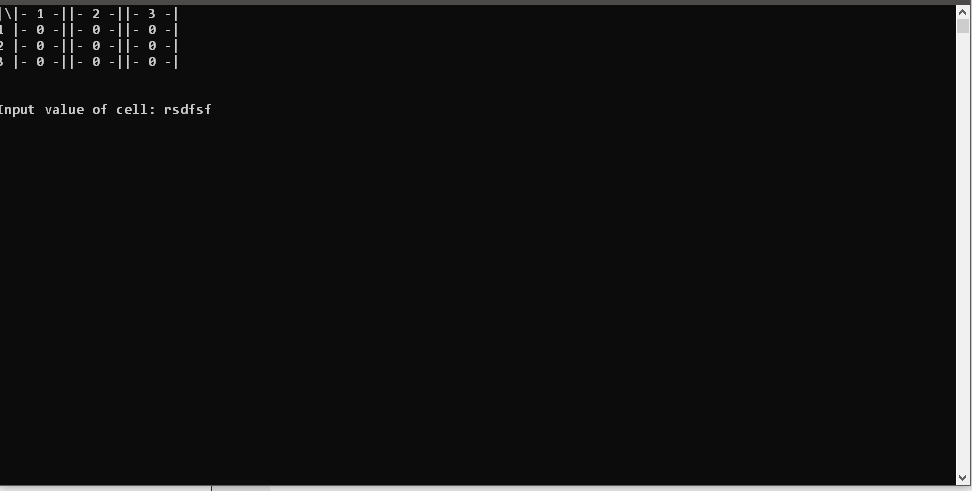
return 0;

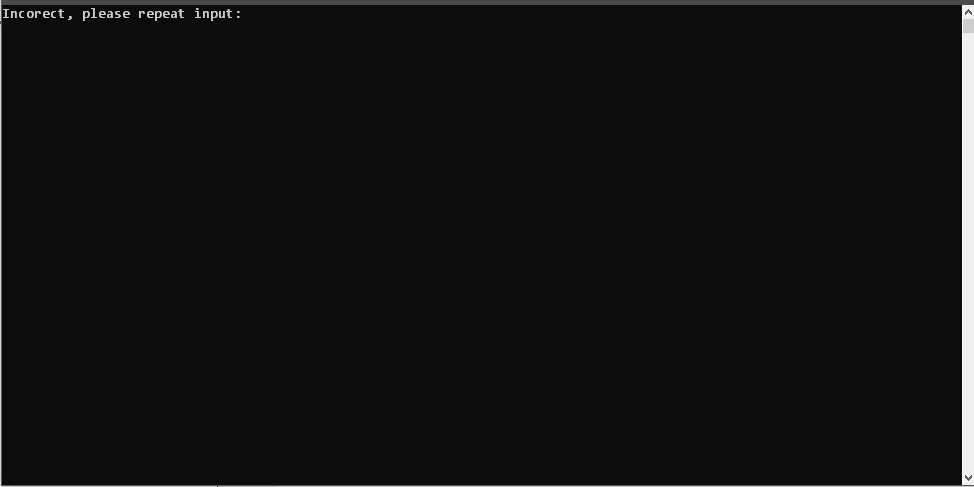
}

## Результат работы программы

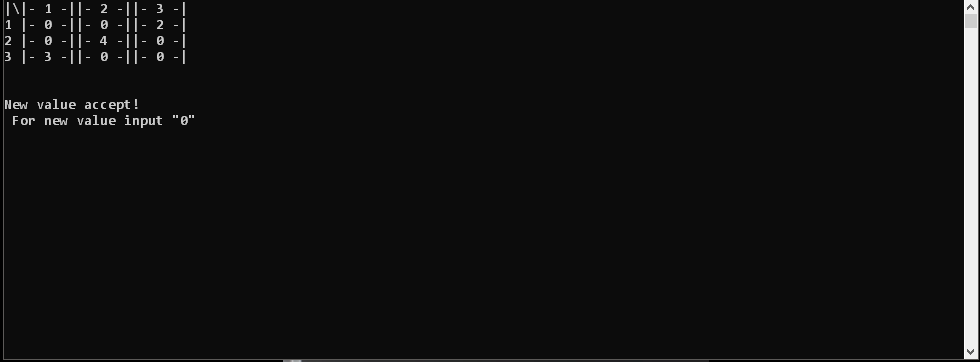


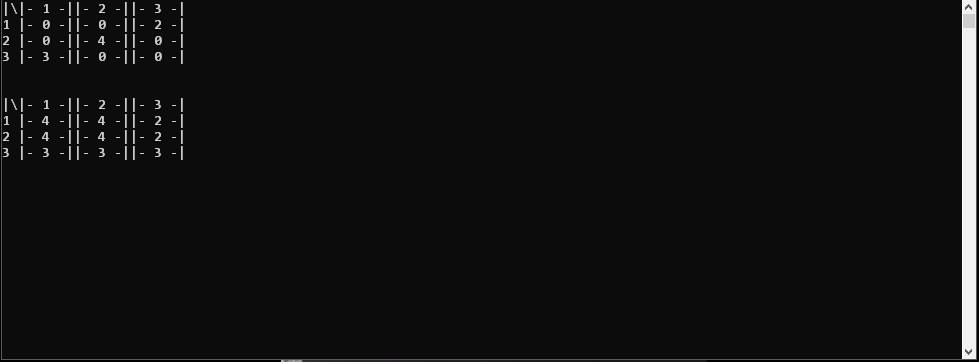
Главное меню





Проверка на ввод символа





Результат работы программы