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

Рекурсия и головоломки

ОТЧЕТ

по лабораторной работе № 4

по дисциплине

Технология программирования

РУКОВОДИТЕЛЬ:

_____ Капранов С. Н.
(подпись) (фамилия, и.,о.)

СТУДЕНТ:

_____ Чеботова М. С.
(подпись) (фамилия, и.,о.)

18-ИСТ-4
(шифр группы)

Работа защищена «___» _____

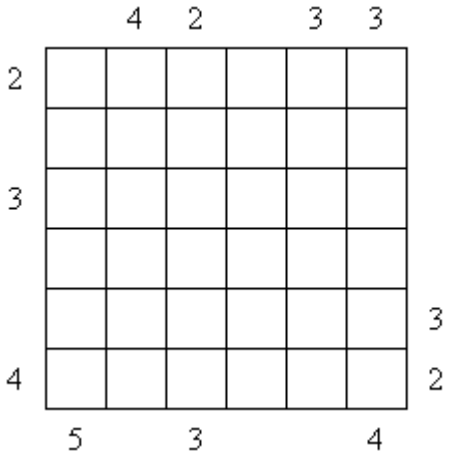
С оценкой _____

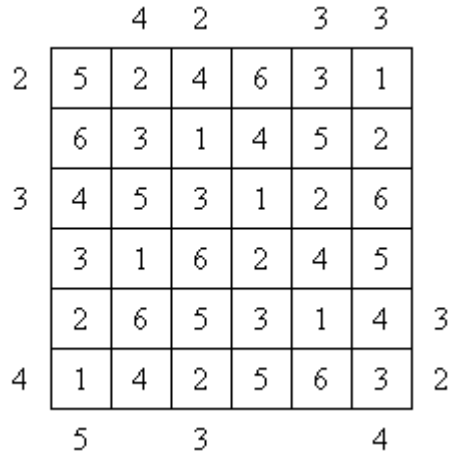
Нижний Новгород
2020

Вариант 20

Задание:

Небоскрёбы (*"Skyscrapers"*) - вид логической головоломки. В ней требуется заполнить числами квадрат так, чтобы в каждой строке и в каждом столбце каждое число использовалось лишь единожды. Заполнение нужно делать с помощью ключевых чисел, считая, что в каждой клетке внутри квадрата стоит дом с количеством этажей, равном стоящему в клетке числу, а числа вне поля означают, сколько домов видно в соответствующей строке или столбце, если смотреть от этого числа.





Листинг программы:

```
#include <iostream>
#include <vector>
#include <ctime>

using namespace std;

//вспомогательные методы для создания класса комбинаций
void swap(vector <int>& a, int i, int j); //поменять местами два числа в векторе
bool NextSet(vector <int>& a, int size); // составить все возможные перестановки

class Combination
{
public:
	vector <int> seq; // последовательность элементов
	int first; //элементов видно сверху или слева
	int second; //элементов видно снизу или справа
	Combination(vector <int> seq)
	{
		this->seq = seq;
		first = second = 0;

		int temp = 0;
		for (auto i = seq.begin(); i != seq.end(); i++)  //подсчет first для комбинации
		{
			if ((*i) > temp)
			{
				temp = (*i);
				first++;
			}
		}

		temp = 0;
		for (auto i = seq.rbegin(); i != seq.rend(); i++) //подсчет second для комбинации
		{
			if ((*i) > temp)
			{
				temp = (*i);
```

```
                        second++;
                }
        }

    }
};


class Board
{
    int size;
    vector <vector <int>> data; //массивых данных поля
    vector <int> outside; // массивы для элементов вне поля (заполнять нужно: верх->лево->право->низ)
    vector <Combination> combinations; //вектор содержащий все возможные комбинации
    bool decided; // ответ на вопрос удалось ли решить задачу?

    void Fill(int row, vector <Combination> combinations); // заполнение поля
    bool HorizontalCheck(int row, vector <Combination>& newcombinations); // проверка в столбце на соответствие
    void ValueSpace(int value); //вспомогательный метод для Print

public:
    Board(int size);
    void Fill()
    {
        int row = 0;
        Fill(row, combinations);
    }
    bool GetDecided() { return decided; }
    void PrintResult(); //сетка для вывода результатов
};

//функция проверки вводимых значений
int checking()
{
    int value = 0;
    while (true)
    {
        cin >> value;
        if (!cin)
        {
            cout << "Nevernyy vvod.Poprobuyte snova" << endl;
            cin.clear();
            while (cin.get() != '\n');
        }
        else
        {
            return value;
        }
    }
}

int main()
{
    setlocale(LC_ALL, "Rus");
    srand(time(NULL));
    cout << "Введите размер поля" << endl;
    int size = checking();
    cin >> size; //ввод размера поля для игры

    Board board(size);

    board.PrintResult(); // вывод пустого поля
    board.Fill();

    if (board.GetDecided())
    {
        board.PrintResult(); //если есть решение, вывод заполненного поля
    }
    else
    {
```

```
		cout << "Нет решения" << endl;
	}
	system("pause");
	return 0;
}

void swap(vector <int>& a, int i, int j)
{
	int s = a[i];
	a[i] = a[j];
	a[j] = s;
}
bool NextSet(vector <int>& a, int size)
{
	int j = size - 2;
	while (j != -1 && a[j] >= a[j + 1]) j--;
	if (j == -1)
		return false; // больше перестановок нет
	int k = size - 1;
	while (a[j] >= a[k]) k--;
	swap(a, j, k);
	int l = j + 1, r = size - 1; // сортируем оставшуюся часть последовательности
	while (l < r)
		swap(a, l++, r--);
	return true;
}

Board::Board(int size)
{
	this->size = size;
	decided = 0;

	data.resize(size);
	for (int i = 0; i < size; i++)
	{
		data[i].resize(size);
	}
	outside.resize(4 * size, 0);

	// заполнение комбинаций
	vector <int> temp;
	for (int i = 0; i < size; i++)
	{
		temp.push_back(i + 1);
	}
	combinations.push_back(temp);
	while (NextSet(temp, size))
	{
		combinations.push_back(temp);
	}

	//заполнение чисел вне поля
	cout << "Введите последовательно числа для клеток вне поля (верхние->левые->правые->нижние)" <<
endl;
	for (int i = 0; i < 4 * size; i++)
	{
		int temp;
		cin >> temp;
		outside[i] = temp;
	}
}
// проверка в столбце
bool Board::HorizontalCheck(int row, vector <Combination>& newcombinations)
{
	int column = 0;
	for (int i = 0; i < size; i++) //выполняем для каждого столбца
	{
		vector <int> temp;
		for (int j = 0; j < row; j++) //записываем начало каждого столбца
		{
			temp.push_back(data[j][i]);
		}
```

```
		auto it = combinations.begin();

		vector <int> temp_for_comb; //массив для возможных следующий элементов в столбце
		//перебор всех комбинаций
		for (it; it != combinations.end(); it++)
		{
			int j = 0;
			for (j; j < row; j++)  //отбрасывание комбинаций если они не соответствуют пермым
элементам в столбце
			{
				if (it->seq[j] != temp[j])
				{
					break;
				}
			}
			// проверка на видимость
			if (j == row)
			{
				if ((it->first == outside[column] || outside[column] == 0) && (it->second ==
outside[size * 3 + column] || outside[size * 3 + column] == 0))
				{
					temp_for_comb.push_back(it->seq[row]);
				}
			}
		}
		//отбрасывание комбинаций которые невозможны для следующего ряда
		for (int c = 0, size = newcombinations.size(); c < size; c++)
		{
			auto j = temp_for_comb.begin();
			for (j; j != temp_for_comb.end(); j++)
			{
				if (*j == newcombinations[c].seq[column])
				{
					break;
				}
			}
			if (j == temp_for_comb.end())
			{
				auto it = newcombinations.begin();
				it += c;
				newcombinations.erase(it);
				c--;
				size--;
			}
		}
		//если нет возможных комбинаций для следующего ряда
		if (newcombinations.size() == 0)
		{
			return 0;
		}
		column++;
	}

	return 1;
}
// заполнение поля
void Board::Fill(int row, vector <Combination> combinations)
{
	if (row == size) //если поле заполнено, проверяем вертикали
	{
		int column = 0;
		for (int i = 0; i < size; i++)
		{
			vector <int> temp;
			for (int j = 0; j < size; j++)
			{
				temp.push_back(data[j][i]);
			}

			auto it = combinations.begin();
```

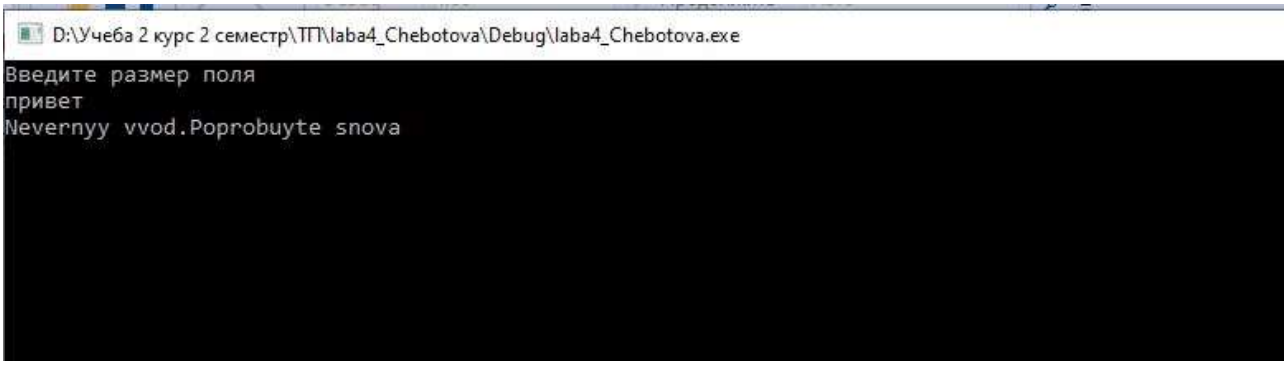
```
                    for (it; it != combinations.end(); it++)
                    {
                           if (it->seq == temp)
                           {
                                  if ((it->first == outside[column] || outside[column] == 0) && (it-
>second == outside[size * 3 + column] || outside[size * 3 + column] == 0))
                                  {
                                         break;
                                  }
                                  else
                                  {
                                         return;
                                  }
                           }
                    }
                    column++;
                    if (it == combinations.end())
                    {
                           return;
                    }
             }
             decided = true;
             return;
       }
       //заполняем текущую строку
       for (auto i : combinations)
       {
             if ((i.first == outside[size + row] || outside[size + row] == 0) && (i.second ==
outside[size * 2 + row] || outside[size * 2 + row] == 0)) //проверка соответствия числам вне поля
             {
                    data[row] = i.seq;
                    row++;

                    vector <Combination> newcombinations = this->combinations;

                    if (row != size)
                    {
                           if (!HorizontalCheck(row, newcombinations)) //проверям горизонтали
                           {
                                  row--;
                                  continue; //если не сошлись горизонтали - меняем строку
                           }
                    }
                    Fill(row, newcombinations); //заполняем следующую строку рекурсивно
                    if (decided) { return; }
                    row--;
             }

       }
}

void Board::PrintResult()
{
       cout << "    ";
       for (int i = 0; i < size; i++)
       {
             if (outside[i] == 0)
             {
                    cout << "     ";
             }
             else
             {
                    cout << " ";
                    ValueSpace(outside[i]);
             }
       }
       cout << endl;

       for (int j = 0; j < size; j++)
       {
             cout << "    ";
             for (int i = 0; i < size; i++)
```

```
		{
			cout << " ----";
		}
		cout << endl;

		if (outside[size + j] == 0)
		{
			cout << "    ";
		}
		else
		{
			ValueSpace(outside[size + j]);
		}

		for (int i = 0; i < size; i++)
		{
			if (data[j][i] == 0)
			{
				cout << "|    ";
			}
			else
			{
				cout << "|";
				ValueSpace(data[j][i]);
			}
		}

		if (outside[size * 2 + j] == 0)
		{
			cout << "|";
		}
		else
		{
			cout << "|";
			ValueSpace(outside[size * 2 + j]);
		}
		cout << endl;


	}
	cout << "    ";
	for (int i = 0; i < size; i++)
	{
		cout << " ----";
	}
	cout << endl;

	cout << "    ";
	for (int i = size * 3; i < size * 4; i++)
	{
		if (outside[i] == 0)
		{
			cout << "     ";
		}
		else
		{
			cout << " ";
			ValueSpace(outside[i]);
		}
	}
	cout << endl;

}
//вспомогательный метод для Print
void Board::ValueSpace(int value)
{
	if (value / 1000)
	{
		cout << value;
	}
	else if (value / 100)
	{
```

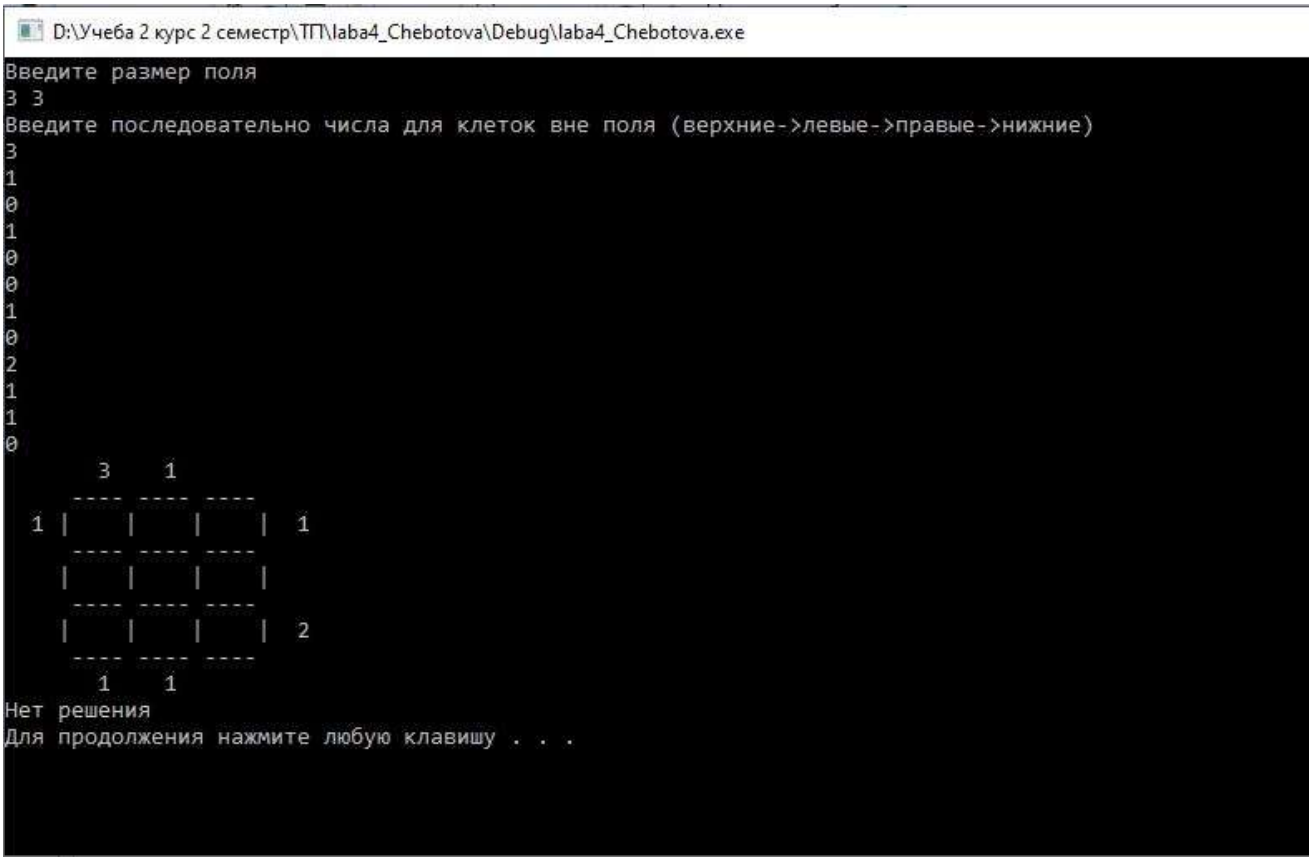
```
			cout << " " << value;
		}
		else if (value / 10)
		{
			cout << " " << value << " ";
		}
		else
		{
			cout << "  " << value << " ";
		}
}
```

Работа программы:

1)Ввод неправильного размера поля



2)Ввод чисел вне поля, для которых нет решения



3) Ввод чисел с решением