ФГБОУ ВО «КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

КАФЕДРА ЮНЕСКО ПО ИНФОРМАЦИОННЫМ ВЫЧИСЛИТЕЛЬНЫМ ТЕХНОЛОГИЯМ

ЛАБОРАТОРНАЯ РАБОТА № 2

по дисциплине «Языки программирования»

Семестр 2

Тема: Динамические многомерные массивы. Игра в жизнь

Выполнил:

студент группы ФИТ-194

Максименко Р.В.

Кемерово, 2020

Задача 1. «Игра жизнь» (6 баллов). Напишите программу, реализующую задачу клеточный автомат “игра в жизнь”. Многие биологические и физические системы можно смоделировать в виде набора объектов, которые с течением времени циклически взаимодействуют и развиваются. Некоторые простейшие системы можно моделировать с помощью клеточных автоматов. Основная идея разделить пространство физической или биологической задачи на отдельные клетки. Каждая клетка – это конечный автомат. После инициализации все клетки сначала совершают один переход в новое состояние, затем второй переход и т.д. Результат каждого перехода зависит от текущего состояния клетки и ее соседей. Дано двухмерное поле клеток. Каждая клетка либо содержит организм (жива), либо пуста (мертва). Каждая клетка имеет восемь соседей, которые расположены сверху, снизу, слева, справа и по четырем диагоналям от нее. Игра “жизнь” происходит следующим образом. Сначала поле инициализируется: определяются мертвые и живые клетки (для этой цели в программе можно использовать генератор случайных чисел). Затем каждая клетка проверяет состояние свое и своих соседей и изменяет свое состояние в соответствии со следующими правилами: a. живая клетка, возле которой меньше двух живых клеток, умирает от одиночества; b. живая клетка, возле которой есть две или три живые клетки, выживает еще на одно поколение; c. живая клетка, возле которой находится больше трех живых клеток, умирает от перенаселения; 5 d. мертвая клетка, рядом с которой есть ровно три живых соседа, оживает. Этот процесс повторяется заданное число шагов (поколений)

#include "pch.h"

#include <iostream>

#include <cstdlib>

#include <cstdio>

#include <ctime>

#include <random>

#include <Windows.h>

using namespace std;

void pole(int\*\* a, int n, int m);

int lifepoints(int\*\* a, int n, int m);

void nbPoints(int nb[][2], int x, int y);

int lifeNBpoints(int\*\* a, int n, int m, int x, int y);

void NewGeneration(int\*\* a, int\*\* a1, int n, int m);

void CopyLastPole(int\*\* a, int\*\* a1, int n, int m);

int Sravnenie(int\*\* a, int\*\* a1, int n, int m);

int main()

{

setlocale(LC\_ALL, "Russian");

int n, m;

cout << "vvedite n="; cin >> n;

cout << "vvedite m="; cin >> m;

int\*\* a;

a = new int\*[n];

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

a[i] = new int[m];

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(1, 10000);

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

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

{

int num = dis(gen);

if (num % 2 == 0)

a[i][j] = 1;

else

a[i][j] = 0;

}

int\*\* PreviousField;

PreviousField = new int\*[n];

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

PreviousField[i] = new int[m];

int livingpoints = -1;

bool ideal = false;

do

{

pole(a, n, m);

CopyLastPole(a, PreviousField, n, m);

NewGeneration(a, PreviousField, n, m);

ideal = Sravnenie(a, PreviousField, n, m) == 0;

livingpoints = lifepoints(a, n, m);

if (ideal)

cout << "Win\n";

if (livingpoints == 0)

cout << "GameOver\n";

Sleep(2000);

} while (livingpoints != 0 || ideal);

return 0;

}

void pole(int\*\* a, int n, int m)

{

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

{

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

{

if (a[i][j] == 1)

cout << '\*';

else

cout << ' ';

cout << ' ';

}

cout << endl;

}

cout << "------------------" << endl;

}

int lifepoints(int\*\* a, int n, int m)

{

int count = 0;

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

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

if (a[i][j] == 1)

count++;

return count;

}

void nbPoints(int nb[][2], int x, int y)

{

int k = 0;

for (int i = x - 1; i <= x + 1; i++)

for (int j = y - 1; j <= y + 1; j++)

{

if (i == x && j == y)

continue;

nb[k][0] = i;

nb[k][1] = j;

k++;

}

}

int lifeNBpoints(int\*\* a, int n, int m, int x, int y)

{

int count = 0;

int nb[8][2];

int x\_, y\_;

nbPoints(nb, x, y);

for (int i = 0; i < 8; i++)

{

x\_ = nb[i][0];

y\_ = nb[i][1];

if (x\_ < 0 || y\_ < 0)

continue;

if (x\_ >= n || y\_ >= m)

continue;

if (a[x\_][y\_] == 1)

count++;

}

return count;

}

void NewGeneration(int\*\* a, int\*\* a1, int n, int m)

{

int livingNB, p;

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

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

{

p = a1[i][j];

livingNB = lifeNBpoints(a1, n, m, i, j);

if (p == 0)

{

if (livingNB == 3)

a[i][j] = 1;

}

else

if (livingNB < 2 || livingNB > 3)

a[i][j] = 0;

}

}

void CopyLastPole(int\*\* a, int\*\* a1, int n, int m)

{

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

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

a1[i][j] = a[i][j];

}

int Sravnenie(int\*\* a, int\*\* a1, int n, int m)

{

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

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

if (a != a1)

return -1;

return 0;

}



