Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Факультет Вычислительной математики и кибернетики

Отчёт по лабораторной работе

Реализация динамической структуры данных типа «очередь» на основе кольцевого буфера памяти   
и её практическое применение для отображения одномерных клеточных автоматов

Выполнил:  
студент ф-та ВМК гр. 82-01  
Савичев М. Р.

Проверил:  
ассистент каф. МО ЭВМ, ВМК  
Кустикова В. Д.

Нижний Новгород  
2012

Оглавление

[Введение 4](#_Toc344142290)

[Постановка задачи 5](#_Toc344142291)

[Руководство пользователя 6](#_Toc344142292)

[Руководство программиста 8](#_Toc344142293)

[Описание структуры программы 8](#_Toc344142294)

[Описание структур данных 9](#_Toc344142295)

[Описание классов 10](#_Toc344142296)

[Описание алгоритмов 13](#_Toc344142297)

[Способ вычисления нового состояния клетки 13](#_Toc344142298)

[Реализация очереди на основе кольцевого буфера 13](#_Toc344142299)

[Алгоритм использования очереди 13](#_Toc344142300)

[Заключение 15](#_Toc344142301)

[Литература 16](#_Toc344142302)

[Приложение. Исходные коды 17](#_Toc344142303)

[Приложение А. Исходный файл Draw.cpp 17](#_Toc344142304)

[Приложение Б. Модуль main.cpp 20](#_Toc344142305)

[Приложение В. Заголовочный файл Configuration.h 21](#_Toc344142306)

[Приложение Г. Модуль Configuration.cpp 21](#_Toc344142307)

[Приложение Д. Заголовочный файл CellAuto.h 21](#_Toc344142308)

[Приложение Е. Модуль CellAuto.cpp 22](#_Toc344142309)

[Приложение Ж. Заголовочный файл CellAuto1D.h 22](#_Toc344142310)

[Приложение З. Модуль CellAuto1D.cpp 23](#_Toc344142311)

[Приложение И. Заголовочный файл TDataCom.h 24](#_Toc344142312)

[Приложение К. Модуль TDataCom.cpp 24](#_Toc344142313)

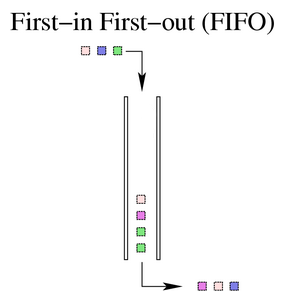
[Приложение Л. Заголовочный файл TData.h 25](#_Toc344142314)

[Приложение М. Модуль TData.cpp 25](#_Toc344142315)

# Введение

Очередь в программировании, как и очередь в магазине, имеет начало и конец. Если приходит новый элемент, то он становится в конец очереди, если необходимо взять элемент из очереди, то он берется из ее начала.

Очередь – структура данных с принципом «первый пришел – первый вышел» (FIFO, first in – first out).



1. Принцип FIFO

Клеточный автомат — дискретная модель, изучаемая в математике, теории вычислимости, физике, теоретической биологии и микромеханике. Включает решётку ячеек, каждая из которых может находиться в одном из конечного множества состояний, таких как 1 и 0. Решетка может быть любой размерности. Для каждой ячейки определено множество ячеек, называемых окрестностью. К примеру, окрестность может быть определено как все ячейки вокруг от текущей. Для работы клеточного автомата требуется задание начального состояния всех ячеек, и правил перехода ячеек из одного состояния в другое. На каждой итерации, используя правила перехода и состояния соседних ячеек, определяется новое состояние каждой ячейки. Обычно правила перехода одинаковы для всех ячеек и применяются сразу ко всей решётке.

# Постановка задачи

Целью данной лабораторной работы является разработка программной системы, которая визуализирует поведение одномерного клеточного автомата с заданной размерностью с помощью библиотеки компьютерного зрения «OpenCV». На экране в текущий момент отображать заданное число конфигураций клеточного автомата. Причем следующая конфигурация появляется снизу, сдвигая предыдущие, а самая старая исчезает. Реализовать последнее с помощью типа данных «очередь». Разработать три способа задания начальной конфигурации: тестовая, случайная (с помощью генератора псевдослучайных чисел) и пользовательская (кликом мышкой). Включить в программу выбор окрестности клеточного автомата, по которой будет пересчитываться следующая конфигурация.

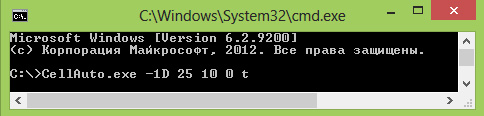
Исходные данные:

1. n – размерность клеточного автомата
2. m – размер очереди
3. Тип окрестности:  
    0 – Неймана  
   C:\Users\Михаил\Desktop\1.jpg  
   1 – Мвона  
   C:\Users\Михаил\Desktop\2.jpg
4. Начальная конфигурация:   
   t – тестовая  
   r – случайная  
   p – пользовательская

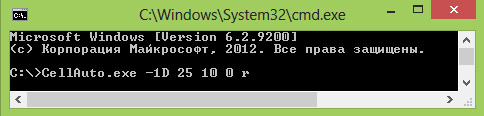
# Руководство пользователя

Для того, чтобы воспользоваться программой необходимо запустить из консоли исполняемый файл «CellAuto.exe» с параметрами, которые являются исходными данными.

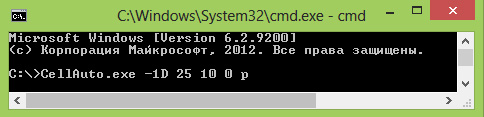
На рис. 2, рис. 3, рис. 4 показан запуск программы с окрестностью Неймана и с тестовой, случайной и пользовательской конфигурациями соответственно.



1. Запуск клеточного автомата с размерностью 25, размером очереди 10, окрестностью Неймана и тестовой конфигурацией



1. Запуск клеточного автомата с размерностью 25, размером очереди 10, окрестностью Неймана и случайной конфигурацией

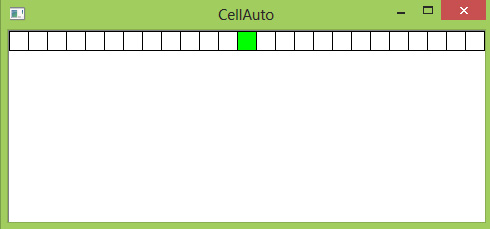


1. Запуск клеточного автомата с размерностью 25, размером очереди 10, окрестностью Неймана и пользовательской конфигурацией

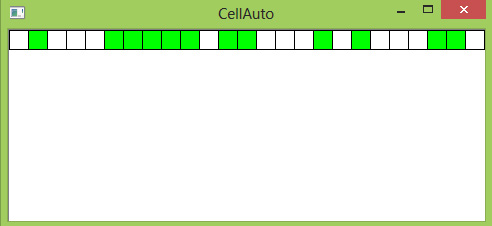
Для того, чтобы задать окрестность Мвона достаточно поменять четвертый параметр на «1».

После нажатия клавиши «Enter» запуститься программа, и, если все параметры будут верны, на экране отобразится тестовая или случайная конфигурация. На рис. 5 и рис. 6 показаны тестовая и возможная случайная конфигурации соответственно.

В случае пользовательской конфигурации необходимо будет с помощью кликов мышки по ячейкам автомата задать начальную конфигурацию и подтвердить её нажатием клавиши «Enter».



1. Тестовая начальная конфигурация



1. Случайная начальная конфигурация

При вводе неверных параметров программа выдаст сообщение о неверных параметрах: «Incorrect parameters».

Далее по нажатию клавиши «Enter» запустится слайд-шоу, нажатие клавиши «Space» останавливает слайд-шоу. При повторном нажатии будет появляться одна следующая конфигурация. Клавиша «Enter» продолжает слайд-шоу.

Выход из программы осуществляется нажатием клавиши «Esc».

# Руководство программиста

## Описание структуры программы

Данная программа разрабатывалась в среде Visual Studio. Решение состоит из трех проектов:

* CellAuto – основная программа
* CellAutoLib – статическая библиотека
* TestApp – тестовая программа

Основная программа использует статическую библиотеку CellAutoLib, которая содержит все объявления и реализации классов, и библиотеку компьютерного зрения OpenCV для визуализации клеточного автомата на экране.

Тестовая программа, используя Google Testing Framework, тестирует методы классов из библиотеки CellAutoLib.

CellAutoLib состоит из пяти модулей:

* Configuration – класс конфигурации клеточного автомата (КА);
* CellAuto – абстрактный класс КА;
* CellAuto1D – классы одномерного КА (ОКА):
  + CellAuto1D – абстрактный класс ОКА;
  + CellAuto1DNeumann – класс ОКА с окрестностью Неймана;
  + CellAuto1DMvon - класс ОКА с окрестностью Мвона;
* TDataCom – класс ошибок структуры данных;
* TData – классы структуры данных:
  + TDataRoot – абстрактный класс структуры данных
  + TDataStack – класс стека
  + TDataQueue – класс очереди

CellAuto состоит из двух модулей: Draw.cpp и main.cpp. Первый содержит функции, которые, работая с классами из CellAutoLib и с OpenCV, выполняют визуализацию клеточного автомата.

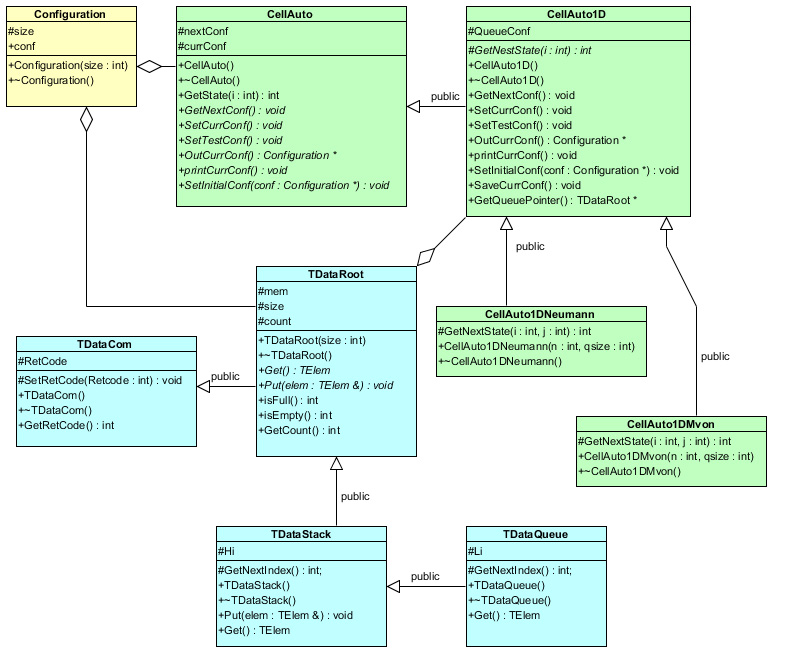
Состав модуля Draw.cpp:

* void RandomConfiguring(Configuration\* conf, int n, int m) – функция создания случайной конфигурации КА;
* void onMouseClick1D(int eventId, int x, int y, int flags, void \*userdata) – функция, которая вызывается при клике мышкой;
* void LinesFor1D(int n, int k) – функция отрисовки линий;
* int PersonalConfiguring1D(int n, int qsize, Configuration\* conf) – функция формирования пользовательской конфигурации КА;
* void Draw1D(CellAuto1D\* object, int n, int qsize) – функция отрисовки КА;
* void ConfiguringAndDrawing1D(int n, int OptionSurround, int OptionConf, int qsize) – функция, которая конфигурирует и отрисовывает КА.

Модуль main.cpp cодержит основную функцию, которая разбирает полученные параметры, проверяет их на корректность, а потом вызывает функцию ConfiguringAndDrawing1D, которая использует все остальные функции из Draw.cpp.

## Описание структур данных

Схема наследования классов в проекте CellAutoLib (рис. 7):



1. Схема наследования классов в библиотеке CellAutoLib

### Описание классов

1. Класс Configuration

|  |  |
| --- | --- |
| Поле | Описание |
| int size | Размерность массива с конфигурацией |
| int \*conf | Указатель на массив с конфигурацией |

|  |  |
| --- | --- |
| Метод | Описание |
| Configuration(int) | Конструктор с параметром – размерностью |
| ~Confifuration() | Деструктор |

1. Класс CellAuto

|  |  |
| --- | --- |
| Поле | Описание |
| Configuration \*nextCurr | Указатель на следующую конфигурацию |
| Configuration \*currConf | Указатель на текущую конфигурацию |

|  |  |
| --- | --- |
| Метод | Описание |
| CellAuto() | Конструктор |
| virtual ~CellAuto() | Деструктор |
| int GetState(int) | Возвращает значение ячейки, входной параметр – номер клетки. |
| void GetNextConf()=0 | Генерирует следующую конфигурацию, используя текущую |
| void SetCurrConf()=0 | Устанавливает следующую конфигурацию как текущую |
| void SetTestConf()=0 | Устанавливает тестовую конфигурацию |
| Configuration\* OutCurrConf()=0 | Возвращает указатель на текущую конфигурацию |
| void printCurrConf()=0 | Печатает текущую конфигурацию в консоли |
| void SetInitialConf(Configuration\*)=0 | Устанавливает переданную конфигурацию в текущую |

1. Класс CellAuto1D

|  |  |
| --- | --- |
| Поле | Описание |
| TDataRoot \*QueueConf | Очередь конфигураций |
| int n | Размерность |

|  |  |
| --- | --- |
| Метод | Описание |
| CellAuto1D(int, int) | Конструктор с параметрами: размерность КА, размер очереди |
| ~CellAuto1D() | Деструктор |
| virtual int GetNextState(int)=0 | Возвращает следующее состояние клетки, входной параметр – номер клетки |
| void SaveCurrConf() | Сохраняет текущую конфигурацию в очередь |
| TDataRoot\* GetQueuePointer() | Возвращает указатель на очередь |

1. Класс CellAuto1DNeumann

|  |  |
| --- | --- |
| Метод | Описание |
| CellAuto1DNeumann(int, int) | Конструктор с параметрами: размерность КА, размер очереди |
| ~CellAuto1DNeumann() | Деструктор |

1. Класс CellAuto1DMvon

|  |  |
| --- | --- |
| Метод | Описание |
| CellAuto1DMvon (int, int) | Конструктор с параметрами: размерность КА, размер очереди |
| ~CellAuto1DMvon () | Деструктор |

1. Класс TDataCom

|  |  |
| --- | --- |
| Поле | Описание |
| RetCode | Код состояния данных |

|  |  |
| --- | --- |
| Метод | Описание |
| void SetRetCode(int) | Устанавливает код состояния, входной параметр – код |
| int GetRetCode() | Возвращает код состояния |
| TDataCom() | Конструктор |
| virtual ~TDataCom() | Деструктор |

1. Класс TDataRoot

|  |  |
| --- | --- |
| Поле | Описание |
| TElem \*mem | Указатель на данные |
| int size | Размер структуры данных |
| int count | Количество элементов в структуре данных |

|  |  |
| --- | --- |
| Метод | Описание |
| TDataRoot(int) | Конструктор, входной параметр – размер структуры данных |
| virtual ~TDataRoot() | Декструктор |
| virtual void Put(const TElem&)=0 | Кладет элемент, входной параметр - элемент |
| virtual TElem Get()=0 | Возвращает элемент |
| int isFull() | Проверка на полноту |
| int isEmpty() | Проверка на пустоту |

1. Класс TStack

|  |  |
| --- | --- |
| Поле | Описание |
| int Hi | Индекс верхнего элемента |

|  |  |
| --- | --- |
| Метод | Описание |
| TStack(int) | Конструктор, входной параметр – размер стека |
| virtual ~TStack() | Деструктор |
| virtual int GetNextIndex() | Возвращает индекс следующей ячейки |

1. Класс TQueue

|  |  |
| --- | --- |
| Поле | Описание |
| int Li | Индекс нижнего элемента |

|  |  |
| --- | --- |
| Метод | Описание |
| TQueue(int) | Конструктор, входной параметр – размер очереди |
| virtual ~TQueue() | Декструктор |

Если в приведенных выше таблицах в каком-то классе не был описан метод, то, значит, он был описан выше, в классе предка. «=0» после прототипа функции означает, что функция чисто виртуальная и реализация находится в классе-наследнике.

## Описание алгоритмов

Большинство функций, реализованных в процессе выполнения лабораторной работы, являются простыми, алгоритм которых ясен сразу же из их описания. Поэтому в этом разделе будут описаны наиболее сложные алгоритмы.

### Способ вычисления нового состояния клетки

Следующее состояние клетки вычисляется с помощью её текущей окрестности. Возникает проблема, когда клетка является граничной. Чтобы решить эту проблему, мы «заворачиваем» решетку в «тор».

Следующее состояние i-й клетки с окрестностью Неймана вычисляется по следующей формуле: (conf[(i-1+n)%n]+conf[(i+1)%n])%2.

### Реализация очереди

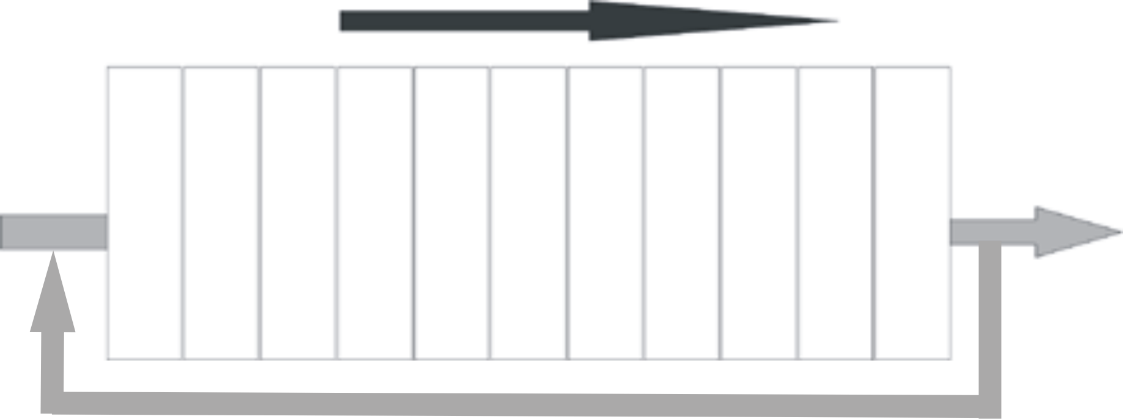
В случае стека вполне понятно, как работают эти функции. Функция Put увеличивает верхний индекс и кладет по новому индексу элемент. Функция Get возвращает элемент по верхнему индексу, затем уменьшает его.

Однако в случае очереди возникает проблема, когда индекс верхнего элемента доходит до значения размера очереди, но это вовсе не значит, что очередь полна, так как могут быть свободные ячейки снизу. Для этого и был введен нижний индекс.

Так же в этих классах была введена функция int GetNextIndex(int), которая в случае стека просто увеличивает индекс на единицу, а в случае очереди увеличивает на единицу и берет по модулю размера очереди. Это сделано для того, чтобы верхний и нижний индексы «перескакивали» вниз.

### Алгоритм использования очереди как кольцевой буфер данных

На рис. 8 показан принцип использования кольцевого буфера.



1. Использование кольцевой очереди

Данный алгоритм используется в функции Draw1D в модуле Draw.cpp для отрисовки текущего содержимого очереди.

Идея проста: достаем элемент из очереди, используем его (рисуем, в нашем случае), потом кладем в очередь. Эти действия должны выполняться в цикле с количеством итерации равным количеству элементов в очереди.

Таким образом мы использовали все элементы из очереди, не изменяя саму очередь.

# Заключение

В ходе выполнения лабораторной работы была разработана программная система, позволяющая визуализировать поведение одномерного клеточного автомата, причем отображая конфигурации в виде очереди с задаваемым размером. Проведенные тесты, которые заключались во введении различных параметров, не выявили утечек памяти и других ошибок в реализации.

# Литература

1. *Страуструп Б*. Программирование: принципы и практика использования C++.: Пер. с англ. – М.: ООО «И. Д. Вильямс», 2011.
2. *Наумов Л., Шалыто А*. Клеточные автоматы. Реализация и эксперименты. Статья опубликована в журнале «Мир ПК», 2003, №8, с. 64-71.
3. Структура данных типа очередь [<http://ru.wikipedia.org/wiki/Очередь_(программирование)>].

# Приложение. Исходные коды

## Приложение А. Исходный файл Draw.cpp

#include "CellAuto1D.h"

#include "TData.h"

#include <iostream>

#include <opencv2/opencv.hpp>

#include <time.h>

using namespace cv;

using namespace std;

#define ESC\_KEY 27

#define ENTER\_KEY 13

#define SPACE\_KEY 32

#define DELAY 100

int width=500, height=500;

const char\* winName = "CellAuto";

const int lineThick=1;

Mat image;

struct DataForMouseClick

{

int n;

int m;

Configuration\* conf;

};

void RandomConfiguring(Configuration\* conf, int n, int m)

{

srand(time(NULL));

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

{

conf->conf[i]=rand()%2;

}

}

void onMouseClick1D(int eventId, int x, int y, int flags, void \*userdata)

{

if (eventId != CV\_EVENT\_LBUTTONDOWN) return;

DataForMouseClick\* data=(DataForMouseClick\*)userdata;

int widthCell=(width-(data->n+1)\*lineThick)/data->n;

int xLTAngle = (x / (widthCell + lineThick)) \* (widthCell + lineThick) + lineThick;

if (y>widthCell+2\*lineThick) return;

if (data->conf->conf[x/(widthCell+lineThick)]==0)

{

rectangle(image, cvRect(xLTAngle, 1, widthCell, widthCell), CV\_RGB(0, 100, 0), CV\_FILLED);

data->conf->conf[x/(widthCell+lineThick)]=1;

}

else

{

rectangle(image, cvRect(xLTAngle, 1, widthCell, widthCell), CV\_RGB(255, 255, 255), CV\_FILLED);

data->conf->conf[x/(widthCell+lineThick)]=0;

}

imshow(winName, image);

}

void LinesFor1D(int n, int k)

{

int widthCell=(width-(n+1)\*lineThick)/n;

int begin = 0;

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

{

begin = i \* (widthCell + lineThick);

line(image, cvPoint(begin, k\*(widthCell+lineThick)+lineThick),

cvPoint(begin, k\*(widthCell+lineThick)+widthCell), CV\_RGB(0, 0, 0),

lineThick);

}

line(image, cvPoint(0, k\*(widthCell+lineThick)), cvPoint(width, k\*(widthCell+lineThick)), CV\_RGB(0, 0, 0),

lineThick);

line(image, cvPoint(0, (k+1)\*(widthCell+lineThick)), cvPoint(width, (k+1)\*(widthCell+lineThick)), CV\_RGB(0, 0, 0),

lineThick);

}

int PersonalConfiguring1D(int n, int qsize, Configuration\* conf)

{

int key=0;

namedWindow(winName,1);

DataForMouseClick data;

data.n=n;

data.m=qsize;

data.conf=conf;

int widthCell=(width-(n+1)\*lineThick)/n;

int xLTAngle=0;

int yLTAngle=0;

cvSetMouseCallback(winName, onMouseClick1D, (void\*)(&data));

width=widthCell\*n+lineThick\*(n+1);

height=widthCell\*qsize+lineThick\*(qsize+1);

image.create(height,width,CV\_32FC3);

rectangle(image, cvRect(0, 0, width, height), CV\_RGB(255, 255, 255), CV\_FILLED);

LinesFor1D(n,0);

imshow(winName,image);

while (key!=ENTER\_KEY && key!=ESC\_KEY && key!=SPACE\_KEY)

{

key=waitKey();

}

if (key==ESC\_KEY) { image.release(); destroyAllWindows(); return -1; }

return 0;

rectangle(image, cvRect(0, 0, width, height), CV\_RGB(255, 255, 255), CV\_FILLED);

}

void Draw1D(CellAuto1D\* object, int n, int qsize)

{

int key=0;

namedWindow(winName,1);

int widthCell=(width-(n+1)\*lineThick)/n;

int xLTAngle=0;

int yLTAngle=0;

TDataRoot\*pQueue=object->GetQueuePointer();

bool show=false;

Configuration\* tmp;

width=widthCell\*n+lineThick\*(n+1);

height=widthCell\*qsize+lineThick\*(qsize+1);

image.create(height,width,CV\_32FC3);

rectangle(image, cvRect(0, 0, width, height), CV\_RGB(255, 255, 255), CV\_FILLED);

while (key!=ESC\_KEY)

{

switch (key)

{

case ENTER\_KEY: { show=true; break; }

case SPACE\_KEY: { show=false; break; }

default: { break; }

}

if (pQueue->isFull()==DataFull) rectangle(image, cvRect(0, 0, width, height), CV\_RGB(255, 255, 255), CV\_FILLED);

object->SaveCurrConf();

for(int i=0; i<pQueue->GetCount(); i++)

{

tmp=pQueue->Get();

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

{

if (tmp->conf[j]==1)

{

xLTAngle = j\*(widthCell + lineThick) + lineThick;

yLTAngle = i\*(widthCell + lineThick) + lineThick;

rectangle(image, cvRect(xLTAngle, yLTAngle, widthCell, widthCell), CV\_RGB(0, 100, 0), CV\_FILLED);

}

}

LinesFor1D(n, i);

pQueue->Put(tmp);

}

object->GetNextConf();

object->SetCurrConf();

imshow(winName,image);

if (show==true) key=waitKey(DELAY);

if (show==false) key=waitKey();

}

image.release();

destroyAllWindows();

}

void ConfiguringAndDrawing1D(int n, int OptionSurround, int OptionConf, int qsize)

{

Configuration conf(n);

CellAuto1D \*object;

switch (OptionSurround)

{

case 0:

{

object=new CellAuto1DNeumann(n, qsize);

break;

}

case 1:

{

object=new CellAuto1DMvon(n, qsize);

break;

}

default:

{

return;

}

}

switch (OptionConf)

{

case 0:

{

object->SetTestConf();

break;

}

case 1:

{

RandomConfiguring(&conf, n, 1);

object->SetInitialConf(&conf);

break;

}

case 2:

{

int tmp=PersonalConfiguring1D(n, qsize, &conf);

if (tmp==-1) { delete object; return; }

object->SetInitialConf(&conf);

break;

}

default:

{

delete object;

return;

}

}

Draw1D(object, n, qsize);

delete object;

*}*

## Приложение Б. Модуль main.cpp

#include <iostream>

#include <conio.h>

#include "CellAuto2D.h"

#include "CellAuto1D.h"

using namespace std;

#define WAITKEY \_getch();

void ConfiguringAndDrawing1D(int, int, int, int);

void ConfiguringAndDrawing2D(int, int, int, int, char\*);

int main(int argc, char \*argv[])

{

if (argc!=6)

{

cout << "Incorrect parameters\n";

WAITKEY

return -1;

}

int n=atoi(argv[2]);

int m=atoi(argv[3]);

if (n<=0 || m<=0)

{

cout << "Incorrect parameters\n";

WAITKEY

return -1;

}

int Surround=atoi(argv[4]);

int OptionConf=-1;

if (strcmp(argv[5], "t")==0) OptionConf=0;

if (strcmp(argv[5], "r")==0) OptionConf=1;

if (strcmp(argv[5], "p")==0) OptionConf=2;

if (OptionConf==-1)

{

cout << "Incorrect parameters\n";

WAITKEY

return -1;

}

if (strcmp(argv[1], "-1D")==0)

{

if (Surround<0 || Surround>1)

{

cout << "Incorrect parameters\n";

WAITKEY

return -1;

}

ConfiguringAndDrawing1D(n, Surround, OptionConf, m);

}

if (strcmp(argv[1], "-2D")==0)

{

if (Surround<0 || Surround>3)

{

cout << "Incorrect parameters\n";

WAITKEY

return -1;

}

char \*formula=0;

if (Surround==3)

{

formula=new char[100];

cout << " A\n";

cout << " E F G\n";

cout << "D L M H B\n";

cout << " K J I\n";

cout << " C\n";

cout << "Please, enter the formula using symbols A-M and +, -, \*, /, (, ):\n";

cin >> formula;

cout << endl;

int i=0;

while (formula[i]!='\0')

{

if ((formula[i]<65 || formula[i]>77) && formula[i]!='+' &&

formula[i]!='-' && formula[i]!='\*' && formula[i]!='/' && formula[i]!='(' && formula[i]!=')')

{

cout << "Incorrect formula\n";

WAITKEY

return -1;

}

i++;

}

}

ConfiguringAndDrawing2D(n, m, Surround, OptionConf, formula);

}

return 0;

}

## Приложение В. Заголовочный файл Configuration.h

#ifndef \_\_CONFIGURATION\_H\_\_

#define \_\_CONFIGURATION\_H\_\_

class Configuration

{

protected:

int size;

public:

Configuration(int);

~Configuration();

int \*conf;

};

#endif

## Приложение Г. Модуль Configuration.cpp

#include <iostream>

using namespace std;

#include "Configuration.h"

Configuration::Configuration(int \_size)

{

size=\_size;

conf=new int[size];

memset(conf, 0, sizeof(int)\*size);

}

Configuration::~Configuration()

{

delete[] conf;

}

## Приложение Д. Заголовочный файл CellAuto.h

#ifndef \_\_CELLAUTO\_H\_\_

#define \_\_CELLAUTO\_H\_\_

#include "Configuration.h"

class CellAuto

{

protected:

Configuration \*nextConf, \*currConf;

public:

CellAuto();

virtual ~CellAuto();

int GetState(int);

virtual void SetCurrConf()=0;

virtual Configuration\* OutCurrConf()=0;

virtual void SetInitialConf(Configuration\*)=0;

virtual void GetNextConf()=0;

virtual void SetTestConf()=0;

virtual void printCurrConf()=0;

};

#endif

## Приложение Е. Модуль CellAuto.cpp

#include <iostream>

using namespace std;

#include "CellAuto.h"

CellAuto::CellAuto()

{

currConf=0;

nextConf=0;

}

CellAuto::~CellAuto()

{

currConf=0;

nextConf=0;

}

int CellAuto::GetState(int k)

{

return currConf->conf[k];

}

## Приложение Ж. Заголовочный файл CellAuto1D.h

#ifndef \_\_CELLAUTO1D\_H\_\_

#define \_\_CELLAUTO1D\_H\_\_

#include "CellAuto.h"

#include "TData.h"

class CellAuto1D: public CellAuto

{

protected:

int n;

virtual int GetNextState(int)=0;

TDataRoot \*QueueConf;

public:

CellAuto1D(int, int);

virtual ~CellAuto1D();

virtual void GetNextConf();

virtual void SetCurrConf();

virtual void SetTestConf();

virtual void printCurrConf();

virtual void SetInitialConf(Configuration\*);

virtual Configuration\* OutCurrConf();

void SaveCurrConf();

TDataRoot\* GetQueuePointer();

};

class CellAuto1DNeumann: public CellAuto1D

{

protected:

virtual int GetNextState(int);

public:

CellAuto1DNeumann(int, int);

~CellAuto1DNeumann();

};

class CellAuto1DMvon: public CellAuto1D

{

protected:

virtual int GetNextState(int);

public:

CellAuto1DMvon(int, int);

~CellAuto1DMvon();

};

#endif

## Приложение З. Модуль CellAuto1D.cpp

#include <iostream>

using namespace std;

#include "CellAuto1D.h"

CellAuto1D::CellAuto1D(int \_n, int \_qsize)

{

n=\_n;

currConf=new Configuration(n);

nextConf=new Configuration(n);

QueueConf=new TQueue(\_qsize);

}

CellAuto1D::~CellAuto1D()

{

n=0;

delete currConf;

delete nextConf;

for(int i=0; i<QueueConf->GetCount(); i++)

delete QueueConf->Get();

}

void CellAuto1D::GetNextConf()

{

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

nextConf->conf[i]=GetNextState(i);

}

void CellAuto1D::SetTestConf()

{

currConf->conf[n/2]=1;

//currConf->conf[n/2+2]=1;

//currConf->conf[n/2-2]=1;

}

void CellAuto1D::printCurrConf()

{

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

cout << currConf->conf[i] << ' ';

cout << endl;

}

void CellAuto1D::SetInitialConf(Configuration\* conf)

{

memcpy(currConf->conf, conf->conf, sizeof(int)\*n);

}

int CellAuto1DNeumann::GetNextState(int k)

{

return (currConf->conf[(k-1+n)%n]+currConf->conf[(k+1)%n])%2;

}

CellAuto1DNeumann::CellAuto1DNeumann(int \_n, int \_qsize)

:CellAuto1D(\_n, \_qsize)

{

}

CellAuto1DNeumann::~CellAuto1DNeumann()

{

}

int CellAuto1DMvon::GetNextState(int k)

{

return (currConf->conf[(k-1+n)%n]+currConf->conf[(k+1)%n]+

currConf->conf[(k-2+n)%n]+currConf->conf[(k+2)%n])%2;

}

CellAuto1DMvon::CellAuto1DMvon(int \_n, int \_qsize)

:CellAuto1D(\_n, \_qsize)

{

}

CellAuto1DMvon::~CellAuto1DMvon()

{

}

Configuration\* CellAuto1D::OutCurrConf()

{

Configuration\* tmp=new Configuration(n);

memcpy(tmp->conf, currConf->conf, sizeof(int)\*n);

return tmp;

}

void CellAuto1D::SetCurrConf()

{

Configuration \*tmp;

tmp=currConf;

currConf=nextConf;

nextConf=tmp;

}

void CellAuto1D::SaveCurrConf()

{

Configuration\* tmp=new Configuration(n);

memcpy(tmp->conf, currConf->conf, sizeof(int)\*n);

if (QueueConf->isFull()) delete QueueConf->Get();

QueueConf->Put(tmp);

}

TDataRoot\* CellAuto1D::GetQueuePointer()

{

return QueueConf;

}

## Приложение И. Заголовочный файл TDataCom.h

#ifndef \_\_TDATACOM\_H\_\_

#define \_\_TDATACOM\_H\_\_

#define DataOK 0

#define DataFull 1

#define DataEmpty 2

class TDataCom

{

protected:

int RetCode;

void SetRetCode(int);

public:

TDataCom();

virtual ~TDataCom();

int GetRetCode();

};

#endif

## Приложение К. Модуль TDataCom.cpp

#include "TDataCom.h"

TDataCom::TDataCom()

{

RetCode=DataEmpty;

}

TDataCom::~TDataCom()

{

RetCode=DataEmpty;

}

int TDataCom::GetRetCode()

{

return RetCode;

}

void TDataCom::SetRetCode(int code)

{

RetCode=code;

}

## Приложение Л. Заголовочный файл TData.h

#ifndef \_\_TDATA\_H\_\_

#define \_\_TDATA\_H\_\_

#include "Configuration.h"

#include "TDataCom.h"

typedef Configuration\* TElem;

class TDataRoot:public TDataCom

{

protected:

TElem \*mem;

int size;

int count;

public:

TDataRoot(int);

virtual ~TDataRoot();

virtual TElem Get()=0;

virtual void Put(const TElem&)=0;

virtual int isFull();

virtual int isEmpty();

int GetCount();

};

class TStack:public TDataRoot

{

protected:

int Hi;

virtual int GetNextIndex(int);

public:

TStack(int);

virtual ~TStack();

virtual void Put(const TElem&);

virtual TElem Get();

};

class TQueue:public TStack

{

protected:

int Li;

virtual int GetNextIndex(int);

public:

TQueue(int);

virtual ~TQueue();

virtual TElem Get();

};

#endif

## Приложение М. Модуль TData.cpp

#include <iostream>

using namespace std;

#include "TData.h"

TDataRoot::TDataRoot(int \_size)

{

size=\_size;

mem=new TElem[size];

count=0;

}

TDataRoot::~TDataRoot()

{

size=0;

delete mem;

count=0;

}

int TDataRoot::isFull()

{

if (size==count)

{

SetRetCode(DataFull);

return DataFull;

}

return DataOK;

}

int TDataRoot::isEmpty()

{

if (count==0)

{

SetRetCode(DataFull);

return DataEmpty;

}

return DataOK;

}

TStack::TStack(int \_size) : TDataRoot(\_size)

{

Hi=-1;

}

TStack::~TStack()

{

Hi=-1;

}

void TStack::Put(const TElem &elem)

{

if (isFull()==DataFull)

{

SetRetCode(DataFull);

return;

}

Hi=GetNextIndex(Hi);

mem[Hi]=elem;

++count;

SetRetCode(DataOK);

}

TElem TStack::Get()

{

if(isEmpty()==DataEmpty)

{

SetRetCode(DataEmpty);

return NULL;

}

TElem temp;

temp=mem[Hi];

Hi--;

count--;

SetRetCode(DataOK);

return temp;

}

int TStack::GetNextIndex(int id)

{

return ++id;

}

TQueue::TQueue(int \_size) : TStack(\_size)

{

Li=0;

}

TQueue::~TQueue()

{

Li=0;

}

int TQueue::GetNextIndex(int id)

{

return (++id)%size;

}

TElem TQueue::Get()

{

if (isEmpty()==DataEmpty)

{

SetRetCode(DataEmpty);

return NULL;

}

TElem temp;

temp=mem[Li];

Li=GetNextIndex(Li);

count--;

SetRetCode(DataOK);

return temp;

}

int TDataRoot::GetCount()

{

return count;

}