MathHarrixLibrary v.3.2.0

А. Б. Сергиенко

26 мая 2013 г.

Аннотация

Библиотека MathHarrixLibrary — это сборник различных математических функций и функций-шаблонов с открытым кодом на языке C++.

Содержание

1 Описание

Сайт: https://github.com/Harrix/MathHarrixLibrary.

Что это такое? Сборник различных математических функций и шаблонов с открытым кодом на языке C++. Упор делается на алгоритмы искусственного интеллекта. Используется только C++.

Что из себя это представляет? Фактически это .cpp и .h файл с исходниками функций и шаблонов, который можно прикрепить к любому проекту на C++. В качестве подключаемых модулей используется только: stdlib.h, time.h, math.h.

Сколько? На данный момент опубликовано функций: **139** (без учета переопределенных функций).

На какие алгоритмы делается упор? Генетические алгоритмы, алгоритмы оптимизации первого порядка и другие системы искусственного интеллекта.

По какой лицензии выпускается? Библиотека распространяется по лицензии Apache License, Version 2.0.

Как найти автора? С автором можно связаться по адресу sergienkoanton@mail.ru или http://vk.com/harrix. Сайт автора, где публикуются последние новости: http://blog.harrix.org, а проекты располагаются по адресу http://harrix.org.

Ваши действия:

- Как установить и пользоваться библиотекой.
- Посмотреть все функции библиотеки. Все функции рассортированы по категориям.
- Читать о случайных числах в библиотеке.
- Как добавить свои новые функции в библиотеку.

2 Установка

Если вы хотите только пользоваться библиотекой, то вам нужна из всего проекта только папки **_library**, в которой располагается собранная библиотека и справка по ней, и папка **demo**, в которой находится программа с демонстрацией работы функций. Все остальные папки вам потребуются, если вы хотите добавлять новые функции.

2.1 Общий алгоритм подключения

- Скопируем себе папку **_library** с готовой последней версией библиотеки на сайте проекта https://github.com/Harrix/MathHarrixLibrary.
- Скопируем файлы **MathHarrixLibrary.cpp**, **MathHarrixLibrary.h** в папку с Вашим проектом на C++.
- Пропишем в проекте:

Код 1. Подключение библиотеки

```
#include "MathHarrixLibrary.h"
```

- Если планируем использовать функции, использующие случайные числа (если не знаем, то тоже сделаем), то в начале программы вызовем:

Код 2. Инициализация генератора случайных чисел

```
MHL_SeedRandom();//Инициализировали генератор случайных чисел
```

- Теперь библиотека готова к работе, и можем ее использовать. Например:

Код 3. Пример использования

```
double x;
x=MHL_RandomNumber();
double degree=MHL_DegToRad(60);
```

2.2 Подключение к Qt на примере Qt 5.0.2

Рассматривается на примере создания Qt Gui Application в Qt 5.0.2 for Desktop (MinGW 4.7) с использованием Qt Creator 2.7.0.

- Скопируем себе папку **_library** с готовой последней версией библиотеки на сайте проекта https://github.com/Harrix/MathHarrixLibrary.
- Скопируем файлы **MathHarrixLibrary.cpp**, **MathHarrixLibrary.h** в папку с Вашим проектом на C++ там, где находится файл проекта *.pro.
- Добавим к проекту файлы **MathHarrixLibrary.cpp** и **MathHarrixLibrary.h**. Для этого по проекту в Qt Creator щелкнем правой кнопкой и вызовем команду **Add Existings Files...**, где выберем наши файлы.
- Пропишем в главном файле исходников проекта **mainwindow.cpp**:

Код 4. Подключение библиотеки

```
#include "MathHarrixLibrary.h"
```

- Если планируем использовать функции, использующие случайные числа (если не знаем, то тоже сделаем), то в начале программы в конструкторе **MainWindow::MainWindow** вызовем:

Код 5. Инициализация генератора случайных чисел

```
MHL_SeedRandom();//Инициализировали генератор случайных чисел
```

То есть получится код:

Код 6. Пример файла mainwindow.cpp с подключенной библиотекой

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include "MathHarrixLibrary.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    MHL_SeedRandom();//Инициализировали генератор случайных чисел
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

- Теперь библиотека готова к работе, и можем ее использовать. Например, добавим textEdit, pushButton и напишем слот кнопки:

Код 7. Пример использования

```
void MainWindow::on_pushButton_clicked()
{
    double x;
    x=MHL_RandomNumber();
    double degree=MHL_DegToRad(60);
}
```

2.3 Подключение к C++ Builder на примере C++ Builder 6.0

- Скопируем себе папку **_library** с готовой последней версией библиотеки на сайте проекта https://github.com/Harrix/MathHarrixLibrary.
- Скопируем файлы **MathHarrixLibrary.cpp**, **MathHarrixLibrary.h** в папку с проектом на C++.
- Пропишем проекте в файле **.cpp** главной формы (часто это **Unit1.cpp**) строчку **#include "MathHarrixLibrary.h"**:

Код 8. Подключение библиотеки

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
```

```
#include "Unit1.h"
#include "MathHarrixLibrary.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
...
```

- Добавим в проект файл **MathHarrixLibrary.cpp** через команду: **Project** → **Add to Project...**.
- Если планируем использовать функции, использующие случайные числа (если не знаем, то тоже сделаем), то в конструкторе главной формы инициализируем генератор случайных чисел:

Код 9. Инициализация генератора случайных чисел

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
MHL_SeedRandom();//Инициализировали генератор случайных чисел
...
}
//---------------------------------------------------------------------------
```

- Теперь библиотека готова к работе, и можем ее использовать. Например, создадим кнопку Button1, текстовое поле Memo1 и в клике на Button1 пропишем:

Код 10. Пример использования

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
double x=MHL_RandomNumber();//получим случайное число
Memo1->Lines->Add("x = "+AnsiString(x));//выведем его
}
//---------------------------------------------------------------------------
```

2.4 Подключение к C++ Builder на примере C++Builder XE4

- Скопируем себе папку **_library** с готовой последней версией библиотеки на сайте проекта https://github.com/Harrix/MathHarrixLibrary.
- Скопируем файлы **MathHarrixLibrary.cpp**, **MathHarrixLibrary.h** в папку с проектом на C++.
- Пропишем проекте в файле **.cpp** главной формы (часто это **Unit1.cpp**) строчку **#include "MathHarrixLibrary.h"**:

Код 11. Подключение библиотеки

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#include "MathHarrixLibrary.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
```

```
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
...
```

- Добавим в проект файл **MathHarrixLibrary.cpp** через команду: **Project** → **Add to Project...**.
- Если планируем использовать функции, использующие случайные числа (если не знаем, то тоже сделаем), то в конструкторе главной формы инициализируем генератор случайных чисел:

Код 12. Инициализация генератора случайных чисел

```
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
MHL_SeedRandom();//Инициализировали генератор случайных чисел
...
}
//---------------------------------------------------------------------------
```

- Теперь библиотека готова к работе, и можем ее использовать. Например, создадим кнопку Button1, текстовое поле Memo1 и в клике на Button1 пропишем:

Код 13. Пример использования

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
double x=MHL_RandomNumber();//получим случайное число
Memo1->Lines->Add("x = "+AnsiString(x));//выведем его
}
//---------------------------------------------------------------------------
```

2.5 Подключение к Microsoft Visual Studio на примере Visual Studio 2012

Используется CLR приложение Windows Forms Application (точнее пустой проект, к которому прикреплена форма) на Visual C++.

- Скопируем себе папку **_library** с готовой последней версией библиотеки на сайте проекта https://github.com/Harrix/MathHarrixLibrary.
- Скопируем файлы **MathHarrixLibrary.cpp**, **MathHarrixLibrary.h** в папку с проектом *.vcxproj на C++.
- Пропишем проекте в файле **.h** главной формы (у меня это **MyForm.h**) строчку **#include "MathHarrixLibrary.h"**:

Код 14. Подключение библиотеки

```
#pragma once
#include "MathHarrixLibrary.h"
...
```

- Добавим в проект файл **MathHarrixLibrary.cpp** через правый клик по проекту: **Добавить** → **Существующий элемент Shift+Alt+A**.

- Если планируем использовать функции, использующие случайные числа (если не знаем, то тоже сделаем), то в конструкторе главной формы инициализируем генератор случайных чисел:

Код 15. Инициализация генератора случайных чисел

```
   public ref class MyForm : public System::Windows::Forms::Form
   {
   public:
      MyForm(void)
      {
         MHL_SeedRandom();//Инициализировали генератор случайных чисел
         InitializeComponent();
         //
         //TODO: добавьте код конструктора
         //
      }
...
```

- Теперь библиотека готова к работе, и можем ее использовать. Например, создадим кнопку button1 и listBox1 и в клике на button1 пропишем:

Код 16. Пример использования

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs
   ^  e) {
      double x=MHL_RandomNumber();//получим случайное число
      listBox1->Items->Add("x = " + x.ToString());//выведем его
       }
```

Как видите, алгоритм подключения почти одинаков.

3 О случайных числах в библиотеке MathHarrixLibrary

Генератор случайных чисел (ГСЧ) — очень важная и нужная функция в программировании. При этом необходим лишь первичный генератор — генератор случайных вещественных чисел в интервале $(0;1)$ по равномерному закону распределения. Все остальные случайные числа с другими законами распределения можно получить из равномерного.

По умолчанию в библиотеке используется стандартный генератор случайных чисел.

Итак, что есть в библиотеке? Есть две функции и одна переменная:

- **MHL_Dummy** — результат инициализации генератора случайных чисел. Значение этой переменной вычисляется автоматически функцией MHL_SeedRandom().
- **MHL_SeedRandom()** — инициализатор генератора случайных чисел. Нужно вызвать один раз за всё время запуска программы, в которой используется библиотека.
- **MHL_RandomNumber()** — непосредственно генератор случайных чисел. В своей реализации использует значение переменной MHL_Dummy.

В файле **MathHarrixLibrary.h** (после объявления констант в начале файла) есть строчки, которые объявляют эти вещи:

Код 17. Объявление функций в MathHarrixLibrary.h

```
//ДЛЯ ГЕНЕРАТОРА СЛУЧАЙНЫХ ЧИСЕЛ
void MHL_SeedRandom(void);//Инициализатор генератора случайных чисел
double MHL_RandomNumber(void);//Генерирует вещественное случайное число из интервала
    (0;1)
```

Код 18. Объявление переменной в MathHarrixLibrary.cpp

```
//ДЛЯ ГЕНЕРАТОРА СЛУЧАЙНЫХ ЧИСЕЛ
unsigned int MHL_Dummy;//Результат инициализации генератора случайных чисел
```

В случае своего желания Вы можете заменить тело функций MHL_SeedRandom() и MHL_RandomNumber() на свои собственные. Ниже представлены варианты, которые предлагаются автором.

Код 19. Стандартный вариант по умолчанию

```
void MHL_SeedRandom(void)
{
/*
Инициализатор генератора случайных чисел.
В данном случае используется самый простой его вариант со всеми его недостатками.
Входные параметры:
 Отсутствуют.
Возвращаемое значение:
 Отсутствуют.
*/
//В качестве начального значения для ГСЧ используем текущее время
MHL_Dummy=(unsigned)time(NULL);
srand(MHL_Dummy);//Стандартная инициализация
rand();//первый вызов для контроля
}
//---------------------------------------------------------------------------
double MHL_RandomNumber(void)
{
```

```
/*
Генератор случайных чисел (ГСЧ).
В данном случае используется самый простой его вариант со всеми его недостатками.
Использовать в функциях по криптографии не стоит.
Входные параметры:
 Отсутствуют.
Возвращаемое значение:
 Случайное вещественное число из интервала (0;1) по равномерному закону распределения
*/
return (double)rand()/(RAND_MAX+1);
}
//---------------------------------------------------------------------------
```

Теперь разберем, как применять данные функции.

- Подключаем библиотеку к Вашему проекту на C++.
- В начале программы **один** раз вызываем функцию MHL_SeedRandom(). Ниже приведены примеры, где обычно стоит вызывать эту функцию.

Код 20. Применение MHL_SeedRandom для консольного приложения

```
int main(void)
{
MHL_SeedRandom();//Инициализировали генератор случайных чисел
...
}
```

Код 21. Применение MHL_SeedRandom для C++Builder

```
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
MHL_SeedRandom();//Инициализировали генератор случайных чисел
...
}
```

Код 22. Применение MHL_SeedRandom для Qt

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    MHL_SeedRandom();//Инициализировали генератор случайных чисел
...
}
```

- Теперь в любом месте программы мы можем получить случайное число из интервала $(0;1)$. Например:

Код 23. Применение ГСЧ

```
double x;
x=MHL_RandomNumber();
```

Результат вызова функции, например: $x = 0,420933187007904$.

Вы можете заменить код этих функций (MHL_SeedRandom, MHL_RandomNumber) на свой генератор случайных чисел в интервале $(0;1)$. При этом работоспособность библиотеки не нарушится.

4 Как добавлять новые функции в библиотеку

Данная глава предназначена для тех, кто хочет добавлять в библиотеку новые функции и развивать данный продукт.

Ваши действия:

- Шаг 0. Прочитать некоторую справочную информацию.
- Шаг 1. Написать и проверить свою функцию в папке **source_demo**.
- Шаг 2. Раскидать в функцию по файлам в папке исходников **source_library**.
- Шаг 3. Собрать библиотеку в папке **make**.
- Шаг 4. Раскидать файлы собранной библиотеки из папки **temp_library** по папкам библиотеки и перекомпилировать некоторые программы и справки.

Шаг 0. Справочная информация.

Вначале надо сориентироваться в структуре библиотеки:

- **_library** — основная папка, в которой распологается готовая библиотека и данная справка;
- **demo** — папка, в которой находится программа DemoMathHarrixLibrary.exe с демонстрацией работы функций;
- **make** — в этой папке находится программа MakeMathHarrixLibrary.exe для собрания готовых файлов библиотеки из исходных материалов из папки source_library. Также там находится справка по этой программе;
- **source_demo** — папка с исходными кодами DemoMathHarrixLibrary.exe из папки demo;
- **source_library** — папка исходных материалов библиотеки. Сами эти файлы библиотекой не являются, так как они потом собираются MakeMathHarrixLibrary.exe;
- **source_make** — папка с исходными кодами MakeMathHarrixLibrary.exe из папки make;
- **LICENSE.txt** и **NOTICE.txt** — файлы Apache лицензии;
- **README.md** — файл информации о проекте в системе GitHub.

Для полноценной работы по добавлению функций вам потребуются:

- программа для проверки работоспособности новых функций и компиляции DemoMathHarrixLibrary.exe (например, Qt 5.0.2 с Qt Creator 2.7.0 или любая другая версия Qt). Для проверки работоспособности библиотеки без компиляции DemoMathHarrixLibrary.exe подойдет любой другой C++ компилятор;
- программа для компиляции *.tex документов в *.pdf для формирования справочных материалов. Автор использует для этого связку MiKTex и TeXstudio (версии MiKTex 2.9 и TeXstudio 2.5.2).

В варианте, который использует автор, в *.tex файлах справкок для отображения русских букв используется модуль **pscyr**. Об его установке можно прочитать в статье http://blog.harrix.org/?p=444.

Далее приведены некоторые спецификации, принятые в данной библиотеке.

- Основу библиотеки составляют функции и шаблоны функций. Имена функций начинаются с **MHL_**, например:

Код 24. Пример названия функции

```
void MHL_NormalizationVectorOne(double *VMHL_ResultVector,int VMHL_N);
```

Имена же шаблонов начинаются с **TMHL_**, например:

Код 25. Пример названия шаблона функции

```
template <class T> int TMHL_SearchFirstZero(T *VMHL_Vector, int VMHL_N);
```

Код функций в итоге будет располагаться в MathHarrixLibrary.cpp, а реализация шаблонов будет располагаться в MathHarrixLibrary.h.

- Количество элементов в одномерном массиве обозначается стандартной переменной **int VMHL_N**.
- Количество элементов в двумерном массиве обозначается стандартными переменными **int VMHL_N** и **int VMHL_M**.
- Возвращаемое значение функций обозначается переменной **VMHL_Result**.
- Возвращаемый вектор (над которым производятся вычисления) обозначается указателем ***VMHL_ResultVector**.
- Возвращаемая матрица (над которой производятся вычисления) обозначается указателем ****VMHL_ResultMatrix**.
- Если функция в качестве параметра имеет одну числовую переменную, то она обозначается **VMHL_X** или **VMHL_X1**. Если есть однотипные переменные, то обозначаются **VMHL_X2** или **VMHL_Y** и так далее.
- Если функция в качестве параметра имеет некий вектор, то он обозначается **VMHL_Vector**.
- Если функция в качестве параметра имеет некую матрицу, то она обозначается **VMHL_Matrix**.
- То есть если входные переменные не имеют какой-то особый смысл, то название переменных стандартно, но в тоже время все входные и выходные переменные могут начинаться с **VMHL_**, чтобы различать их от внутренних, но во отличии от выходных значений это есть **не обязательное условие**.

Далее приведена последовательность действий, которую надо выполнить для добавления новой функции. Допустим мы хотим добавить функцию **double MHL_Func(double VMHL_X)**.

Шаг 1. Вначале нам нужно реализовать саму функцию и проверить ее работоспособность. Если вы хотите работать не через средства, предоставляемые библиотекой, то этот шаг можно пропустить.

- Заходим в папку **source_demo** и открываем проект **DemoMathHarrixLibrary.pro** в Qt Creator.
- Добавляем в конец файлов **MathHarrixLibrary.cpp** и **MathHarrixLibrary.h** функцию, которую хотим добавить. Например, в MathHarrixLibrary.cpp добавляем:

Код 26. Что добавляем в MathHarrixLibrary.cpp

```
int  MHL_Func(int VMHL_X)
{
/*
Умножает число на 2.
Входные параметры:
  x - число, которое будет умножаться.
```

```
Возвращаемое значение:
 Число, умноженное на 2.
*/
  return 2*VMHL_X;
}
```

А в MathHarrixLibrary.h добавляем:

Код 27. Что добавляем в MathHarrixLibrary.h

```
int  MHL_Func(int VMHL_X);
```

Замечание. В .h файл добавляем до строчки «**#endif // MATHHARRIXLIBRARY_H**».
Замечание. Если вы добавляете шаблон функции, то его реализацию надо добавлять в MathHarrixLibrary.h.

- Теперь перейдем в проекте DemoMathHarrixLibrary.pro в файл **mainwindow.cpp**.
- Вначале этого файла идет следующий код:

Код 28. mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>
#include <QFile>
#include <QDesktopServices>
#include <QUrl>
#include <QDir>
#include <QStandardItemModel>

#include "MathHarrixLibrary.h"

#include "QtHarrixLibrary.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    DS=QDir::separator();
    path=QGuiApplication::applicationDirPath()+DS;//путь к папке

    MHL_SeedRandom();//Инициализация датчика случайных чисел

    QStandardItemModel *model = new QStandardItemModel;//новая модель списка
    QStandardItem *item;//элемент списка

    //добавление новых элементов
    item = new QStandardItem(QString("TMHL_FillVector"));
    model->appendRow(item);

   //Сюда нужно добавить код

   ...

    //соединение модели списка с конкретным списком
    ui->listView->setModel(model);

    ui->listView->setEditTriggers(QAbstractItemView::NoEditTriggers);
```

```
}
```

- Там, где написан комментарий «**//Сюда нужно добавить код**» необходимо добавить две строчки:

Код 29. Что добавить в mainwindow.cpp

```
    item = new QStandardItem(QString("[Имя вашей функции]"));
    model->appendRow(item);
```

То есть в рассматриваемом примере вы должны добавить:

Код 30. Что добавить в mainwindow.cpp в примере

```
    item = new QStandardItem(QString("MHL_Func"));
    model->appendRow(item);
```

Добавление данного кода добавить вашу функцию в список, которые будут отображаться в программе при запуске. По сути, удобнее было бы извлекать из обычного текстового файла. Может в будущих версиях так и сделаю, но все равно вам нужно потом писать код демонстрации функции, поэтому занесение в текстовой файл не предусмотрел.

- Далее найдем функцию **MainWindow::on_listView_clicked**:

Код 31. MainWindow::on_listView_clicked

```
void MainWindow::on_listView_clicked(const QModelIndex &index)
{
    Html="<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0//EN\" \"http://www.w3.
       org/TR/REC-html40/strict.dtd\">\n<html><head><meta name=\"qrichtext\"
       content=\"1\" />\n<meta http-equiv=\"Content-Type\" content=\"text/
       html; charset=utf-8\" />\n<style type=\"text/css\">\np, li { white-
       space: pre-wrap; }\n</style></head><body style=\" font-family:'MS
       Shell Dlg 2'; font-size:8.25pt; font-weight:400; font-style:normal
       ;\">\n";

    QString NameFunction;//Какая функция вызывается

    //выдергиваем текст
    NameFunction=index.data(Qt::DisplayRole).toString();

   //Сюда нужно добавить код

   ...

    //Показ итогового результата
    Html+="</body></html>";
    HQt_SaveFile(Html, path+"temp.html");
    ui->webView->setUrl(QUrl::fromLocalFile(path+"temp.html"));
}
```

- Там, где написан комментарий «**//Сюда нужно добавить код**» добавляете код следующего типа:

Код 32. Добавление демонстрации работы функции

```
    if (NameFunction=="[Имя вашей функции]")
    {
   //Реализация демонстрации функции
    }
```

Вместо «**[Имя вашей функции]**» пишите название вашей функции, такое же, что добавляли выше. Вместо комментария «**//Сюда нужно добавить код**» добавьте реализацию демонстрации вашей функции. Например, для рассматриваемого примера код будет выглядеть так:

Код 33. Добавление демонстрации работы функции на примере

```
    if (NameFunction=="MHL_Func")
    {
        int x=5;

        //Вызов функции
        int y=MHL_Func(x);

        //Используем полученный результат
        MHL_ShowNumber (x,"Первоначальное число", "x");
        MHL_ShowNumber (y,"Умноженное число", "y");
        //Первоначальное число:
        //x=5
        //Умноженное число:
        //y=10
    }
```

- Рассмотрим немного этот код. После вызова функции идет комментарий «**//Используем полученный результат**». После него надо вывести в webView нужную информацию. Для этого лучше использовать стандартные функции, список который написан ниже.
- После вывода функций в виде комментариев показывается тот текст, который может продемонстрироваться при вызове функции. У нас это код:

Код 34. Закомментированный результат работы функции

```
        //Первоначальное число:
        //x=5
        //Умноженное число:
        //y=10
```

Теперь рассмотрим какие функции используются для вывода результата. Типичными объектами, над которыми выполняются действия по выводу, являются: числа, вектора, матрицы. Их мы стандартизовано и выводим, используя некоторые функции. Так как библиотека MathHarrixLibrary может использоваться на различных системах C++, а вывод информации в каждой системе может быть разным, то функции вывода строились таким образом, чтобы внешне выглядели однотипно в любой системе C++, так как в справке к функциям из библиотеки функции вывода также будут присутствовать. Итак, использование функций внешне должно быть везде одинаковым для всех систем C++. Поэтому вы можете их переписать под свои нужды.

- **MHL_NumberToText** — функция перевода числа в строку;
- **MHL_ShowNumber** — функция вывода числа;
- **MHL_ShowVector** — функция вывода вектора (одномерного массива);
- **MHL_ShowVectorT** — функция вывода вектора (одномерного массива) в строку одну, то есть это транспонированный вектор;
- **MHL_ShowMatrix** — функция вывода матрицы.

Далее функции рассмотрены подробнее.

- **MHL_ShowNumber** — функция вывода числа.

Код 35. Синтаксис функции MHL_ShowNumber

```
template <class T> void MHL_ShowNumber (T VMHL_X, QString TitleX, QString
    NameX);
```

Входные параметры:

- VMHL_X — выводимое число;
- TitleX — заголовок выводимого числа;
- NameX — обозначение числа.

Пример использования функции:

Код 36. Пример использования MHL_ShowNumber

```
MHL_ShowNumber (x,"Первоначальное число", "x");
//Первоначальное число:
//x=5
```

И для этой функции покажем исходный код:

Код 37. Реализация функции MHL_ShowNumber

```
//mainwindow.cpp
template <class T> void MainWindow::MHL_ShowNumber (T VMHL_X, QString TitleX,
    QString NameX)
{
    /*
    Функция выводит число VMHL_X в textEdit.
    Входные параметры:
     VMHL_X - выводимое число;
     TitleX - заголовок выводимого числа;
     NameX - обозначение числа.
    Возвращаемое значение:
     Отсутствует.
    */
    QString VMHL_Result;
    VMHL_Result=THQt_ShowNumber (VMHL_X, TitleX, NameX);// из QtHarrixLibrary.
        h
    Html+=VMHL_Result;
}
//-------------------------------------------------------------------------

//QtHarrixLibrary.h
template <class T> QString THQt_ShowNumber (T VMHL_X, QString TitleX, QString
    NameX)
{
    /*
    Функция возвращает строку с выводом некоторого числа VMHL_X с HTML кодами.
        Для добавление в html файл.
    Входные параметры:
     VMHL_X - выводимое число;
     TitleX - заголовок выводимого числа;
     NameX - обозначение числа.
    Возвращаемое значение:
     Строка с HTML кодами с выводимым числом.
    */
    QString VMHL_Result;

    VMHL_Result="<p><b>"+TitleX+":</b><br>";
```

```
    VMHL_Result+=NameX+"=<b><font color=\"#4200ff\">"+QString::number(VMHL_X)+
       "</font></b></p>\n";

    return VMHL_Result;
}
//---------------------------------------------------------------------------
```

В функции MainWindow::on_listView_clicked() есть еще код для сохранения и вывода значения переменной Html в виде *.html файла.
В предыдущей версии библиотеки для программы демонстрации работы функций использовалась система C++Builder 6. Там эта функции реализовывалась так:

Код 38. Реализация функции MHL_ShowNumber в C++Builder 6

```
template <class T> void MHL_ShowNumber (T X, AnsiString A, AnsiString B)
{
Form1->Memo1->Lines->Add(A+":");
Form1->Memo1->Lines->Add(B+" = "+AnsiString(X));
Form1->Memo1->Lines->Add("");
}
//---------------------------------------------------------------------------
```

Как видим, вид функций по внешнему виду однотипен — различается только тип строк, который используется.

- **MHL_NumberToText** — выводит число в строку.

Код 39. Синтаксис функции MHL_NumberToText

```
template <class T> QString MainWindow::MHL_NumberToText (T VMHL_X);
```

Входные параметры:
- VMHL_X — выводимое число.

Пример использования функции:

Код 40. Пример использования MHL_NumberToText

```
MHL_ShowNumber(Deg,"Угол "+MHL_NumberToText(Rad)+" радиан","равен в градусах")
    ;
//Угол 3.14159 радиан:
//равен в градусах=180
```

- **MHL_ShowVector** — функция вывода вектора (одномерного массива).

Код 41. Синтаксис функции MHL_ShowVector

```
template <class T> void MHL_ShowVector (T *VMHL_Vector, int VMHL_N, QString
    TitleVector, QString NameVector);
```

Входные параметры:
- Vector — указатель на выводимый вектор;
- VMHL_N — количество элементов вектора a;
- TitleVector — заголовок выводимого вектора;
- NameVector — обозначение вектора.

Пример использования функции:

Код 42. Пример использования MHL_ShowVector

```
MHL_ShowVector (a,VMHL_N,"Заполненный вектор", "a");
//Заполненный вектор:
```

```
//a =
//5
//5
//5
//5
//5
//5
//5
//5
//5
//5
```

- **MHL_ShowVectorT** — функция вывода вектора (одномерного массива) в транспонированном виде, то есть в одну строку.

Код 43. Синтаксис функции MHL_ShowVectorT

```
template <class T> void MHL_ShowVectorT (T *VMHL_Vector, int VMHL_N, QString
    TitleVector, QString NameVector);
```

Входные параметры:

- Vector — указатель на выводимый вектор;
- VMHL_N — количество элементов вектора a;
- TitleVector — заголовок выводимого вектора;
- NameVector — обозначение вектора.

Пример использования функции:

Код 44. Пример использования MHL_ShowVectorT

```
MHL_ShowVector (a,VMHL_N,"Заполненный вектор", "a");
//Заполненный вектор:
//a = 5 5 5 5 5 5 5 5 5 5
```

- **MHL_ShowMatrix** — функция вывода матрицы.

Код 45. Синтаксис функции MHL_ShowMatrix

```
template <class T> void MHL_ShowMatrix (T **VMHL_Matrix, int VMHL_N, int
    VMHL_M, QString TitleMatrix, QString NameMatrix);
```

Входные параметры:

- VMHL_Matrix — указатель на выводимую матрицу;
- VMHL_N — количество строк в матрице;
- VMHL_M — количество столбцов в матрице;
- TitleMatrix — заголовок выводимой матрицы;
- NameMatrix — обозначение матрицы.

Пример использования функции:

Код 46. Пример использования MHL_ShowMatrix

```
        MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Матрица", "x");
        //Матрица:
        //x =
        //0 1  2  3  4
        //1 2  3  4  5
        //2 3  4  5  6
        //3 4  5  6  7
```

```
        //4 5  6  7  8
        //5 6  7  8  9
        //6 7  8  9  10
```

Итак, мы добавили в DemoMathHarrixLibrary.pro нашу функцию и проверили ее работоспособность.

Шаг 2. Теперь нам нужно добавить нашу функцию в исходники. Все исходные материалы располагаются в папке **source_library**. В ней располагаются некоторые файлы, которые нам не особы интересны (подробнее в файле справке к программе MakeMathHarrixLibrary.exe в файле **make\MakeMathHarrixLibrary_Help.pdf**) и папки (например, **Вектора (Одномерные массивы)**). Каждая такая папка является разделом функций в библиотеке. Вам нужно выбрать папку, в которую вы будете добавлять свою функцию или создать свою собственную, если ничто не подходит по смыслу.

Каждая функция или шаблон функции в разделе (выбранной вами папке) предоставляется следующими файлами:

- **<File>.cpp** или **<File>.tpp** — код функции;
- **<File>.h** — заголовочный файл функции;
- **<File>.tex** — справка по функции;
- **<File>.desc** — описание функции;
- **<File>.use** — пример использования функции;
- **<File>_<name>.pdf** — множество рисунков, необходимых для справки по функции (необязательные файлы);
- **<File>_<name>.png** — множество рисунков, необходимых для справки по функции (необязательные файлы);

Без файлов <File>.cpp (или <File>.tpp), <File>.h, <File>.tex, <File>.desc, <File>.use библиотека соберется, но с ошибками, то есть каждая функция должна быть представима минимум 5 файлами (могут быть дополнительно рисунки).

Считаем далее, что вы выбрали папку **<Dir>** в папке source_library.

- Создайте в папке <Dir> текстовой файл **<File>.h**, где <File> — это имя функции, то есть в рассматриваемом примере мы должны создать файл **MHL_Func.h**.
- В файл <File>.h мы добавляем объявление нашей функции, например:

Код 47. Содержимое MHL_Func.h

```
int  MHL_Func(int VMHL_X);
```

- В файл <File>.cpp мы добавляем код нашей функции, например:

Код 48. Содержимое MHL_Func.cpp

```
int  MHL_Func(int VMHL_X)
{
/*
Умножает число на 2.
Входные параметры:
  x - число, которое будет умножаться.
Возвращаемое значение:
 Число, умноженное на 2.
```

```
*/
  return 2*VMHL_X;
}
```

Если у нас не функция, а шаблон функции, то мы создаем файл <File>.tpp (обратите внимание на расширение файла), например:

Код 49. Содержимое TMHL_FillVector.tpp

```
template <class T> void TMHL_FillVector(T *VMHL_ResultVector, int VMHL_N, T x)
{
/*
Функция заполняет вектор значениями, равных x.
Входные параметры:
 VMHL_ResultVector - указатель на преобразуемый массив;
 VMHL_N - количество элементов в массиве;
 x - число, которым заполняется вектор.
Возвращаемое значение:
 Отсутствует.
*/
for (int i=0;i<VMHL_N;i++) VMHL_ResultVector[i]=x;
}
```

- В файл <File>.desc мы добавляем описание нашей функции, например:

Код 50. Содержимое MHL_Func.desc

```
Умножает число на 2.
```

- В файл <File>.tex мы добавляем справку к нашей функции в виде куска tex кода, например:

Код 51. Содержимое MHL_Func.tex

```
\textbf{Входные параметры:}

 x --- входной параметр.

\textbf{Возвращаемое значение:}
Число умноженное на 2.
```

- В файл <File>.use мы добавляем код примера использования функции, например:

Код 52. Содержимое MHL_Func.use

```
int x=5;

//Вызов функции
int y=MHL_Func(x);

//Используем полученный результат
MHL_ShowNumber (x,"Первоначальное число", "x");
MHL_ShowNumber (y,"Умноженное число", "y");
//Первоначальное число:
//x=5
//Умноженное число:
//y=10
```

- Если хотите использовать рисунки в tex справке к функции, то в папку <Dir> скопируйте рисунки вида **<File>_<name>.pdf** и **<File>_<name>.png**

- Если мы используем дополнительную переменную перечисляемого типа, то добавляем ее в файл **Enum.h** в папке **source_library**.
- Если мы хотим использовать глобальную константу, то добавляем ее в файл **Const.h** в папке **source_library**.
- Если мы хотим использовать глобальную переменную, то добавляем ее в файл **AdditionalVariables.cpp** в папке **source_library**.

Замечание. Если вы хотите переопределить функцию какую-нибудь, то вы добавляете переопределенные функции, их объявления в уже существующие файлы, а не создаете новые.

Замечание. Класс и его методы нужно оформлять в одном файле *.cpp, *.h и др., а не разбивать на несколько и прописывать каждый метод в отдельном.

Итак, мы добавили в папку source_library нашу функцию. Теперь нужно перестроить библиотеку и провести замену файлов.

Шаг 3. Сборка библиотеки. Перейдем в папку **make** в корне файлов библиотеки. В ней есть программа MakeMathHarrixLibrary.exe и справка к ней MakeMathHarrixLibrary_Help.pdf.

- Включим программу **MakeMathHarrixLibrary.exe**.
- Нажмем кнопку **Собрать библиотеку**.
- В окне программы будет отчет об собрании библиотеки, например:

Код 53. Пример отчета о сборке библиотеки

```
Начало формирования файлов библиотеки...
Загрузили файл Header.cpp
Загрузили файл AdditionalVariables.cpp
Загрузили файл Random.cpp
Загрузили файл Const.h
Загрузили файл Random.cpp
Загрузили файл Enum.h
Загрузили файл Install.tex
Загрузили файл Random.tex
Загрузили файл Addnew.tex

Было найдено 1 папок - разделов библиотеки

Рассматриваем папку Вектора (Одномерные массивы)
Было найдено 15 файлов в папке

Загрузили файл FuncF.cpp
Загрузили файл FuncF.desc
Загрузили файл FuncF.h
Загрузили файл FuncF.tex
Загрузили файл FuncF.use
Загрузили файл MHL_Func.cpp
Загрузили файл MHL_Func.desc
Загрузили файл MHL_Func.h
Загрузили файл MHL_Func.tex
Загрузили файл MHL_Func.use
Загрузили файл TMHL_FillVector.desc
Загрузили файл TMHL_FillVector.h
Загрузили файл TMHL_FillVector.tex
Загрузили файл TMHL_FillVector.tpp
Загрузили файл TMHL_FillVector.use
Из 15 файлов нужными нам оказалось 15 файлов в папке
```

```
Загрузили файл Description_part2.tex
Загрузили файл Description_part1.tex
Загрузили файл Title.tex

Сохранили файл MathHarrixLibrary.cpp
Сохранили файл MathHarrixLibrary.h
Сохранили файл MathHarrixLibrary_Help.tex

Скопировали файл names.tex
Скопировали файл packages.tex
Скопировали файл styles.tex

Ошибки не были зафиксированы.
Конец формирования файлов библиотеки.
```

Если ошибок нет, то все прошло нормально.

- Также нам будет продемонстрирована папка **temp_library** с сформированными файлами библиотеки.

Итак, мы собрали файлы библиотеки.

Шаг 4. Разберем файлы из папки **temp_library**.

- Скопируем файлы **MathHarrixLibrary.cpp** и **MathHarrixLibrary.h** в папку **_library**.
- Откройте файл **MathHarrixLibrary_Help.tex** в LaTeX программе (автор использует TeXstudio) и скомпилируйте его.
  В итоге в папке temp_library появится файл **MathHarrixLibrary_Help.pdf**. Скопируйте этот файл в папку **_library**.
- Теперь разберемся с программой для демонстрации. Как мы помним, в ней в самом начале мы проверяли свою функцию.
  - Скопируем файлы **MathHarrixLibrary.cpp** и **MathHarrixLibrary.h** в папку **source_demo**.
  - Откройте **DemoMathHarrixLibrary.pro** из папки source_demo в Qt Creator и скомпилирйте приложение (в режиме Release).
  - Найдите папку, в которую скомпилировался проект. Это может быть папка проектов Qt, или папка появится в корневой папке библиотеки MathHarrixLibrary.
  - Скопируйте файл **DemoMathHarrixLibrary.exe** в папку **demo**.
- Удалим папку **temp_library** после всех наших действий.
- Если папка с скомпилированным файлом DemoMathHarrixLibrary.exe появилась в корневой папке библиотеки, то удалите ее (например, build-DemoMathHarrixLibrary-Desktop_Qt_5_0_2_MinGW_32bit-Release).
- Отредактируйте на своё усмотрение файл README.md, где напишите о новых изменениях.
- В файлах **README.md** и **source_library\Title.tex** поменяйте номер версии библиотеки.

Вот, вроде и всё. Мы добавили новую функцию и обновили все файлы и папки библиотеки.

5 Список функций

Вектора (Одномерные массивы)

1. **MHL_EuclidNorma** — Функция вычисляет евклидовую норму вектора.
2. **MHL_NoiseInVector** — Функция добавляет к элементам выборки аддитивную помеху (плюс-минус сколько-то процентов модуля разности минимального и максимального элемента выборки).
3. **TMHL_AcceptanceLimits** — Функция вмещает вектор VMHL_ResultVector в прямоугольную многомерной области, определяемой левыми границами и правыми границами. Если какая-то координата вектора выходит за границу, то значение этой координаты принимает граничное значение.
4. **TMHL_CheckElementInVector** — Функция проверяет наличие элемента a в векторе x.
5. **TMHL_EqualityOfVectors** — Функция проверяет равенство векторов.
6. **TMHL_FibonacciNumbersVector** — Функция заполняет массив числами Фибоначчи.
7. **TMHL_FillVector** — Функция заполняет вектор значениями, равных x.
8. **TMHL_MaximumOfVector** — Функция ищет максимальный элемент в векторе (одномерном массиве).
9. **TMHL_MinimumOfVector** — Функция ищет минимальный элемент в векторе (одномерном массиве).
10. **TMHL_MixingVector** — Функция перемешивает массив. Поочередно рассматриваются номера элементов массивов. С некоторой вероятностью рассматриваемый элемент массива меняется местами со случайным элементом массива.
11. **TMHL_MixingVectorWithConjugateVector** — Функция перемешивает массив вместе со сопряженным массивом. Поочередно рассматриваются номера элементов массивов. С некоторой вероятностью рассматриваемый элемент массива меняется местами со случайным элементом массива. Пары элементов первого массива и сопряженного остаются без изменения.
12. **TMHL_NumberOfDifferentValuesInVector** — Функция подсчитывает число различных значений в векторе (одномерном массиве).
13. **TMHL_NumberOfMaximumOfVector** — Функция ищет номер максимального элемента в векторе (одномерном массиве).
14. **TMHL_NumberOfMinimumOfVector** — Функция ищет номер минимального элемента в векторе (одномерном массиве).
15. **TMHL_NumberOfNegativeValues** — Функция подсчитывает число отрицательных значений в векторе (одномерном массиве).
16. **TMHL_NumberOfPositiveValues** — Функция подсчитывает число положительных значений в векторе (одномерном массиве).
17. **TMHL_NumberOfZeroValues** — Функция подсчитывает число нулевых значений в векторе (одномерном массиве).
18. **TMHL_OrdinalVector** — Функция заполняет вектор значениями, равные номеру элемента, начиная с единицы.

19. **TMHL_OrdinalVectorZero** — Функция заполняет вектор значениями, равные номеру элемента, начиная с нуля.
20. **TMHL_ReverseVector** — Функция меняет порядок элементов в массиве на обратный. Преобразуется подаваемый массив.
21. **TMHL_SearchFirstNotZero** — Функция возвращает номер первого ненулевого элемента массива.
22. **TMHL_SearchFirstZero** — Функция возвращает номер первого нулевого элемента массива.
23. **TMHL_SumSquareVector** — Функция вычисляет сумму квадратов элементов вектора.
24. **TMHL_SumVector** — Функция вычисляет сумму элементов вектора.
25. **TMHL_VectorMinusVector** — Функция вычитает поэлементно из одного массива другой и записывает результат в третий массив. Или в переопределенном виде функция вычитает поэлементно из одного массива другой и записывает результат в первый массив.
26. **TMHL_VectorMultiplyNumber** — Функция умножает вектор на число.
27. **TMHL_VectorPlusVector** — Функция складывает поэлементно из одного массива другой и записывает результат в третий массив. Или в переопределенном виде функция складывает поэлементно из одного массива другой и записывает результат в первый массив.
28. **TMHL_VectorToVector** — Функция копирует содержимое вектора (одномерного массива) в другой.
29. **TMHL_ZeroVector** — Функция зануляет массив.

Геометрия

1. **TMHL_BoolCrossingTwoSegment** — Функция определяет наличие пересечения двух отрезков. Координаты отрезков могут быть перепутаны по порядку в каждом отрезке.

Гиперболические функции

1. **MHL_Cosech** — Функция возвращает гиперболический косеканс.
2. **MHL_Cosh** — Функция возвращает гиперболический косинус.
3. **MHL_Cotanh** — Функция возвращает гиперболический котангенс.
4. **MHL_Sech** — Функция возвращает гиперболический секанс.
5. **MHL_Sinh** — Функция возвращает гиперболический синус.
6. **MHL_Tanh** — Функция возвращает гиперболический тангенс.

Дифференцирование

1. **MHL_CenterDerivative** — Численное значение производной в точке (центральной разностной производной с шагом 2h).
2. **MHL_LeftDerivative** — Численное значение производной в точке (разностная производная влево).
3. **MHL_RightDerivative** — Численное значение производной в точке (разностная производная вправо).

Интегрирование

1. **MHL_IntegralOfRectangle** — Интегрирование по формуле прямоугольников с оценкой точности по правилу Рунге. Считается интеграл функции на отрезке [a,b] с погрешностью порядка Epsilon.
2. **MHL_IntegralOfSimpson** — Интегрирование по формуле Симпсона с оценкой точности по правилу Рунге. Считается интеграл функции на отрезке [a,b] с погрешностью порядка Epsilon.
3. **MHL_IntegralOfTrapezium** — Интегрирование по формуле трапеции с оценкой точности по правилу Рунге. Считается интеграл функции на отрезке [a,b] с погрешностью порядка Epsilon.

Математические функции

1. **MHL_ArithmeticalProgression** — Арифметическая прогрессия. n-ый член последовательности.
2. **MHL_ExpMSxD2** — Функция вычисляет выражение $exp(-x * x/2)$.
3. **MHL_GeometricSeries** — Геометрическая прогрессия. n-ый член последовательности.
4. **MHL_GreatestCommonDivisorEuclid** — Функция находит наибольший общий делитель двух чисел по алгоритму Евклида.
5. **MHL_HowManyPowersOfTwo** — Функция вычисляет, какой минимальной степенью двойки можно покрыть целое положительное число.
6. **MHL_InverseNormalizationNumberAll** — Функция осуществляет обратную нормировку числа из интервала $[0; 1]$ в интервал $[-\infty; \infty]$, которое было осуществлено функцией MHL_NormalizationNumberAll.
7. **MHL_LeastCommonMultipleEuclid** — Функция находит наименьшее общее кратное двух чисел по алгоритму Евклида.
8. **MHL_NormalizationNumberAll** — Функция нормирует число из интервала $[-\infty; \infty]$ в интервал $[0; 1]$. При этом в нуле возвращает 0.5, в $-\infty$ возвращает 0, в ∞ возвращает 1. Если $x < y$, то $MHL\_NormalizationNumberAll(x) < MHL\_NormalizationNumberAll(y)$. Под бесконечностью принимается машинная бесконечность.
9. **MHL_Parity** — Функция проверяет четность целого числа.
10. **MHL_SumGeometricSeries** — Геометрическая прогрессия. Сумма первых n членов.
11. **MHL_SumOfArithmeticalProgression** — Арифметическая прогрессия. Сумма первых n членов.
12. **MHL_SumOfDigits** — Функция подсчитывает сумму цифр любого целого числа.
13. **TMHL_Abs** — Функция возвращает модуль числа.
14. **TMHL_FibonacciNumber** — Функция вычисляет число Фибоначчи, заданного номера.
15. **TMHL_HeavisideFunction** — Функция Хевисайда (функция одной переменной).
16. **TMHL_Max** — Функция возвращает максимальный элемент из двух.
17. **TMHL_Min** — Функция возвращает минимальный элемент из двух.
18. **TMHL_NumberInterchange** — Функция меняет местами значения двух чисел.
19. **TMHL_PowerOf** — Функция возводит произвольное число в целую степень.
20. **TMHL_Sign** — Функция вычисляет знака числа.

21. **TMHL_SignNull** — Функция вычисляет знака числа. При нуле возвращает 1.

Матрицы

1. **TMHL_ColInterchange** — Функция переставляет столбцы матрицы.
2. **TMHL_ColToMatrix** — Функция копирует в матрицу (двумерный массив) из вектора столбец.
3. **TMHL_DeleteColInMatrix** — Функция удаляет k столбец из матрицы (начиная с нуля). Все правостоящие столбцы сдвигаются влево на единицу. Последний столбец зануляется.
4. **TMHL_DeleteRowInMatrix** — Функция удаляет k строку из матрицы (начиная с нуля). Все нижестоящие строки поднимаются на единицу. Последняя строка зануляется.
5. **TMHL_FillMatrix** — Функция заполняет матрицу значениями, равных x.
6. **TMHL_IdentityMatrix** — Функция формирует единичную квадратную матрицу.
7. **TMHL_MatrixMinusMatrix** — Функция вычитает две матрицы. Или для переопределенной варианта функция вычитает два матрицы и результат записывает в первую матрицу.
8. **TMHL_MatrixMultiplyMatrix** — Функция перемножает матрицы.
9. **TMHL_MatrixMultiplyMatrixT** — Функция умножает матрицу на транспонированную матрицу.
10. **TMHL_MatrixMultiplyNumber** — Функция умножает матрицу на число.
11. **TMHL_MatrixPlusMatrix** — Функция суммирует две матрицы. Или для переопределенной варианта функция суммирует два матрицы и результат записывает в первую матрицу.
12. **TMHL_MatrixT** — Функция транспонирует матрицу.
13. **TMHL_MatrixTMultiplyMatrix** — Функция умножает транспонированную матрицу на матрицу.
14. **TMHL_MatrixToCol** — Функция копирует из матрицы (двумерного массива) в вектор столбец.
15. **TMHL_MatrixToMatrix** — Функция копирует содержимое матрицы (двумерного массива) a в массив VMHL_ResultMatrix.
16. **TMHL_MatrixToRow** — Функция копирует из матрицы (двумерного массива) в вектор строку.
17. **TMHL_MaximumOfMatrix** — Функция ищет максимальный элемент в матрице (двумерном массиве).
18. **TMHL_MinimumOfMatrix** — Функция ищет минимальный элемент в матрице (двумерном массиве).
19. **TMHL_MixingRowsInOrder** — Функция меняет строки матрицы в порядке, указанным в массиве b.
20. **TMHL_NumberOfDifferentValuesInMatrix** — Функция подсчитывает число различных значений в матрице.
21. **TMHL_RowInterchange** — Функция переставляет строки матрицы.
22. **TMHL_RowToMatrix** — Функция копирует в матрицу (двумерный массив) из вектора строку.

23. **TMHL_SumMatrix** — Функция вычисляет сумму элементов матрицы.
24. **TMHL_ZeroMatrix** — Функция зануляет матрицу.

Метрика

1. **TMHL_Chebychev** — Функция вычисляет расстояние Чебышева.
2. **TMHL_CityBlock** — Функция вычисляет манхэттенское расстояние между двумя массивами.
3. **TMHL_Euclid** — Функция вычисляет евклидово расстояние.

Оптимизация

1. **MHL_BinaryMonteCarloAlgorithm** — Метод Монте-Карло (Monte-Carlo). Простейший метод оптимизации на бинарных строках. В простонародье его называют "методом научного тыка". Алгоритм оптимизации. Ищет максимум функции пригодности FitnessFunction.

Перевод единиц измерений

1. **MHL_DegToRad** — Функция переводит угол из градусной меры в радианную.
2. **MHL_RadToDeg** — Функция переводит угол из радианной меры в градусную.

Случайные объекты

1. **MHL_BitNumber** — Функция с вероятностью P (или 0.5 в переопределенной функции) возвращает 1. В противном случае возвращает 0.
2. **MHL_RandomRealMatrix** — Функция заполняет матрицу случайными вещественными числами из определенного интервала [Left;Right].
3. **MHL_RandomRealMatrixInCols** — Функция заполняет матрицу случайными вещественными числами из определенного интервала. При этом элементы каждого столбца изменяются в своих пределах.
4. **MHL_RandomRealMatrixInElements** — Функция заполняет матрицу случайными вещественными числами из определенного интервала. При этом каждый элемент изменяется в своих пределах.
5. **MHL_RandomRealMatrixInRows** — Функция заполняет матрицу случайными вещественными числами из определенного интервала. При этом элементы каждой строки изменяются в своих пределах.
6. **MHL_RandomRealVector** — Функция заполняет массив случайными вещественными числами из определенного интервала [Left;Right].
7. **MHL_RandomRealVectorInElements** — Функция заполняет массив случайными вещественными числами из определенного интервала, где на каждую координату свои границы изменения.
8. **MHL_RandomVectorOfProbability** — Функция заполняет вектор случайными значениями вероятностей. Сумма всех элементов вектора равна 1.
9. **TMHL_BernulliVector** — Функция формирует случайный вектор Бернулли.
10. **TMHL_RandomArrangingObjectsIntoBaskets** — Функция предлагает случайный способ расставить N объектов в VMHL_N корзин при условии, что в каждой корзине может располагаться только один предмет.

11. **TMHL_RandomBinaryMatrix** — Функция заполняет матрицу случайно нулями и единицами.
12. **TMHL_RandomBinaryVector** — Функция заполняет вектор (одномерный массив) случайно нулями и единицами.
13. **TMHL_RandomIntMatrix** — Функция заполняет матрицу случайными целыми числами из определенного интервала [n;m).
14. **TMHL_RandomIntMatrixInCols** — Функция заполняет матрицу случайными целыми числами из определенного интервала [n;m). При этом элементы каждого столбца изменяются в своих пределах.
15. **TMHL_RandomIntMatrixInElements** — Функция заполняет матрицу случайными целыми числами из определенного интервала [n;m). При этом каждый элемент изменяется в своих пределах.
16. **TMHL_RandomIntMatrixInRows** — Функция заполняет матрицу случайными целыми числами из определенного интервала [n;m). При этом элементы каждой строки изменяются в своих пределах.
17. **TMHL_RandomIntVector** — Функция заполняет массив случайными целыми числами из определенного интервала [n,m).
18. **TMHL_RandomIntVectorInElements** — Функция заполняет массив случайными целыми числами из определенного интервала [n_i,m_i). При этом для каждого элемента массива свой интервал изменения.

Случайные числа

1. **MHL_RandomNormal** — Случайное число по нормальному закону распределения.
2. **MHL_RandomUniform** — Случайное вещественное число в интервале [a;b] по равномерному закону распределения.
3. **MHL_RandomUniformInt** — Случайное целое число в интервале [n,m) по равномерному закону распределения.

Сортировка

1. **TMHL_BubbleDescendingSort** — Функция сортирует массив в порядке убывания методом "Сортировка пузырьком".
2. **TMHL_BubbleSort** — Функция сортирует массив в порядке возрастания методом "Сортировка пузырьком".
3. **TMHL_BubbleSortInGroups** — Функция сортирует массив в порядке возрастания методом "Сортировка пузырьком"в группах данного массива. Имеется массив. Он делится на группы элементов по m элементов. Первые m элементов принадлежат первой группе, следующие m элементов - следующей и т.д. (Разумеется, в последней группе может и не оказаться m элементов). Потом в каждой группе элементы сортируются по возрастанию.
4. **TMHL_BubbleSortWithConjugateVector** — Функция сортирует массив вместе с сопряженный массивом в порядке возрастания методом "Сортировка пузырьком". Пары элементов первого массива и сопряженного остаются без изменения.
5. **TMHL_BubbleSortWithTwoConjugateVectors** — Функция сортирует массив вместе с двумя сопряженными массивами в порядке возрастания методом "Сортировка пузырьком". Пары элементов первого массива и сопряженного остаются без изменения.

Статистика и теория вероятности

1. **MHL_DensityOfDistributionOfNormalDistribution** — Плотность распределения вероятности нормированного и центрированного нормального распределения.
2. **MHL_DistributionFunctionOfNormalDistribution** — Функция распределения нормированного и центрированного нормального распределения.
3. **MHL_StdDevToVariance** — Функция переводит среднеквадратичное уклонение в значение дисперсии случайной величины.
4. **MHL_VarianceToStdDev** — Функция переводит значение дисперсии случайной величины в среднеквадратичное уклонение.
5. **TMHL_Mean** — Функция вычисляет среднее арифметическое массива.
6. **TMHL_Median** — Функция вычисляет медиану выборки.
7. **TMHL_SampleCovariance** — Функция вычисляет выборочную ковариацию выборки.
8. **TMHL_Variance** — Функция вычисляет выборочную дисперсию выборки.

Тригонометрические функции

1. **MHL_Cos** — Функция возвращает косинус угла в радианах.
2. **MHL_CosDeg** — Функция возвращает косинус угла в градусах.
3. **MHL_Cosec** — Функция возвращает косеканс угла в радианах.
4. **MHL_CosecDeg** — Функция возвращает косеканс угла в градусах.
5. **MHL_Cotan** — Функция возвращает котангенс угла в радианах.
6. **MHL_CotanDeg** — Функция возвращает котангенс угла в градусах.
7. **MHL_Sec** — Функция возвращает секанс угла в радианах.
8. **MHL_SecDeg** — Функция возвращает секанс угла в градусах.
9. **MHL_Sin** — Функция возвращает синус угла в радианах.
10. **MHL_SinDeg** — Функция возвращает синус угла в градусах.
11. **MHL_Tan** — Функция возвращает тангенс угла в радианах.
12. **MHL_TanDeg** — Функция возвращает тангенс угла в градусах.

6 Функции

6.1 Вектора (Одномерные массивы)

6.1.1 MHL_EuclidNorma

Функция вычисляет евклидовую норму вектора.

Код 54. Синтаксис

```
double MHL_EuclidNorma(double *a,int VMHL_N);
```

Входные параметры:

a — указатель на вектор;

VMHL_N — размер массива.

Возвращаемое значение:

Значение евклидовой нормы вектора.

Формула:

$$EuclidNormaVector = \sqrt{\sum_{i=1}^{n} (a_i)^2}.$$

Код 55. Пример использования

```
int VMHL_N=5;//Размер массива
double *x;
x=new double[VMHL_N];
//Заполним случайными числами
MHL_RandomRealVector (x,0,10,VMHL_N);

//Вызов функции
double a=MHL_EuclidNorma(x,VMHL_N);

//Используем полученный результат
MHL_ShowVector (x,VMHL_N,"Вектор", "x");
// Вектор:
//x =
//2.22504
//5.2655
//5.00092
//5.7428
//9.11682

MHL_ShowNumber (a,"Значение евклидовой нормы вектора", "a");
// Значение евклидовой нормы вектора:
// a=13.1826

delete [] x;
```

6.1.2 MHL_NoiseInVector

Функция добавляет к элементам выборки аддитивную помеху (плюс-минус сколько-то процентов модуля разности минимального и максимального элемента выборки).

Код 56. Синтаксис

```
void MHL_NoiseInVector(double *VMHL_ResultVector, double percent, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на массив;

percent — процент шума;

VMHL_N — количество элементов в массивах.

Возвращаемое значение:

Отсутствует.

Формула:

$$b = \frac{percent \cdot (\max x_i - \min x_i)}{100};$$
$$x_i = x_i + random\left(-\frac{b}{2}, \frac{b}{2}\right),$$

где $x_i \in VMHL\_ResultVector$, $i = \overline{1, VMHL\_N}$.

Код 57. Пример использования

```
int VMHL_N=10;//Размер массива
double *x;
x=new double[VMHL_N];
//Заполним массив номерами от 1
TMHL_OrdinalVector(x,VMHL_N);
MHL_ShowVector (x,VMHL_N,"Вектор", "x");
//Вектор:
//x =
//1
//2
//3
//4
//5
//6
//7
//8
//9
//10

double percent=double(MHL_RandomUniformInt(0,100));//Процент помехи

//Вызов функции
MHL_NoiseInVector(x,percent,VMHL_N);

//Используем полученный результат

MHL_ShowNumber (percent,"Процент помехи", "percent");
```

```
        //Процент помехи:
        //percent=89
        MHL_ShowVector (x,VMHL_N,"Вектор с помехой", "x");
        //Вектор с помехой:
        //x =
        //-1.95828
        //2.17942
        //1.76139
        //4.45956
        //3.82128
        //8.0003
        //6.80982
        //5.94739
        //9.03153
        //8.59053

        delete [] x;
```

6.1.3 TMHL_AcceptanceLimits

Функция вмещает вектор VMHL_ResultVector в прямоугольную многомерной области, определяемой левыми границами и правыми границами. Если какая-то координата вектора выходит за границу, то значение этой координаты принимает граничное значение.

Код 58. Синтаксис

```
template <class T> void TMHL_AcceptanceLimits(T *VMHL_ResultVector, T *Left, T *Right
    , int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на вектор (в него же записывается исправленный вектор);

Left — вектор левых границ;

Right — вектор правых границ;

VMHL_N — размерность вектора.

Возвращаемое значение:

Отсутствует.

Код 59. Пример использования

```
        int VMHL_N=10;//Размер массива
        double *a;
        a=new double[VMHL_N];
        double *Left;
        Left=new double[VMHL_N];
        double *Right;
        Right=new double[VMHL_N];
        TMHL_FillVector(Left,VMHL_N,-1.);//Левая граница
        TMHL_FillVector(Right,VMHL_N,1.);//Правая граница

        for (int i=0;i<VMHL_N;i++) a[i]=MHL_RandomUniform(-1.1,1.1);
        MHL_ShowVector (a,VMHL_N,"Вектор", "a");
        //Вектор:
```

```
//a =
//-0.199268
//-1.07664
//-0.395917
//0.170935
//-0.720935
//-1.07878
//1.01608
//-0.594714
//-1.09678
//0.2513

//Вызов функции
TMHL_AcceptanceLimits(a,Left,Right,VMHL_N);

//Используем полученный результат
MHL_ShowVector (Left,VMHL_N,"Левые границы", "Left");
//Левые границы:
//Left =
//-1
//-1
//-1
//-1
//-1
//-1
//-1
//-1
//-1

MHL_ShowVector (Right,VMHL_N,"Правые границы", "Right");
// Правые границы:
//Right =
//1
//1
//1
//1
//1
//1
//1
//1
//1

MHL_ShowVector (a,VMHL_N,"Отредактированный вектор", "a");
//Отредактированный вектор:
//a =
//-0.199268
//-1
//-0.395917
//0.170935
//-0.720935
//-1
//1
//-0.594714
//-1
//0.2513

delete [] a;
delete [] Left;
```

```
        delete [] Right;
```

6.1.4 TMHL_CheckElementInVector

Функция проверяет наличие элемента a в векторе x.

Код 60. Синтаксис

```
template <class T> int TMHL_CheckElementInVector(T *x, int VMHL_N, T a);
```

Входные параметры:

x — указатель на вектор;

VMHL_N — размер массива;

a — проверяемый элемент.

Возвращаемое значение:

Номер (начиная с нуля) первого элемента, равного искомому. Если такого элемента нет, то возвращается -1.

Код 61. Пример использования

```
        int i;
        int VMHL_N=10;//Размер массива (число строк)
        int *a;
        a=new int[VMHL_N];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniformInt(0,5);
        int k=MHL_RandomUniformInt(0,5);//искомое число

        //Вызов функции
        int Search=TMHL_CheckElementInVector(a,VMHL_N,k);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Вектор", "a");
        //Вектор:
        //a =
        //2
        //1
        //2
        //1
        //0
        //1
        //0
        //3
        //0
        //0

        MHL_ShowNumber (k,"Искомое число", "k");
        //Искомое число:
        //k=3

        MHL_ShowNumber (Search,"находится в векторе a под номером", "Search");
        //находится в векторе a под номером:
        //Search=7
```

```
        delete [] a;
```

6.1.5 TMHL_EqualityOfVectors

Функция проверяет равенство векторов.

Код 62. Синтаксис

```
template <class T> bool TMHL_EqualityOfVectors(T *a, T *b, int VMHL_N);
```

Входные параметры:

a — первый вектор;

b — вторый вектор;

VMHL_N - размер векторов.

Возвращаемое значение:

true — вектора совпадают;

false — вектора не совпадают.

Код 63. Пример использования

```
        int VMHL_N=5;//Размер массива (число строк)
        int *a;
        a=new int[VMHL_N];
        int *b;
        b=new int[VMHL_N];

        int x=MHL_RandomUniformInt(0,2);//заполнитель для вектора a
        int y=MHL_RandomUniformInt(0,2);//заполнитель для вектора b
        TMHL_FillVector (a, VMHL_N, x);
        TMHL_FillVector (b, VMHL_N, y);


        //Вызов функции
        int Q=TMHL_EqualityOfVectors(a,b,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Вектор", "a");
        //Вектор:
        //a =
        //1
        //1
        //1
        //1
        //1

        MHL_ShowVector (b,VMHL_N,"Вектор", "b");
        //Вектор:
        //b =
        //0
        //0
        //0
        //0
        //0
```

```
        MHL_ShowNumber (Q,"Равны ли вектора", "Q");
        // Равны ли вектора:
        //Q=0


        delete [] a;
        delete [] b;
```

6.1.6 TMHL_FibonacciNumbersVector

Функция заполняет массив числами Фибоначчи.

Код 64. Синтаксис

```
template <class T> void TMHL_FibonacciNumbersVector(T *VMHL_ResultVector, int VMHL_N)
    ;
```

Входные параметры:

VMHL_ResultVector — указатель на массив, в который записывается результат;

VMHL_N — размер массива.

Возвращаемое значение:

Отсутствует.

Код 65. Пример использования

```
        int VMHL_N=MHL_RandomUniformInt(5,15);//Размер массива
        double *x;
        x=new double[VMHL_N];

        //Вызов функции
        TMHL_FibonacciNumbersVector(x,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (x,VMHL_N,"Вектор, заполненый числами Фибоначи", "x");
        //Вектор, заполненый числами Фибоначи:
        //x =
        //1
        //1
        //2
        //3
        //5
        //8
        //13
        //21
        //34
        //55
        //89
        //144

        delete [] x;
```

6.1.7 TMHL_FillVector

Функция заполняет вектор значениями, равных x.

Код 66. Синтаксис

```
template <class T> void TMHL_FillVector(T *VMHL_ResultVector, int VMHL_N, T x);
```

Входные параметры:

VMHL_ResultVector — указатель на преобразуемый массив;

VMHL_N — количество элементов в массиве;

x — число, которым заполняется вектор.

Возвращаемое значение: Отсутствует.

Код 67. Пример использования

```
int VMHL_N=10;//Размер массива
int *a;
a=new int[VMHL_N];

int x=5;//заполнитель

//Вызов функции
TMHL_FillVector(a,VMHL_N,x);

//Используем полученный результат
// MHL_ShowVector (a,VMHL_N,"Заполненный вектор", "a");
//Заполненный вектор:
// a[0] = 5
// a[1] = 5
// a[2] = 5
// a[3] = 5
// a[4] = 5
// a[5] = 5
// a[6] = 5
// a[7] = 5
// a[8] = 5
// a[9] = 5
delete [] a;
```

6.1.8 TMHL_MaximumOfVector

Функция ищет максимальный элемент в векторе (одномерном массиве).

Код 68. Синтаксис

```
template <class T> T TMHL_MaximumOfVector(T *VMHL_Vector, int VMHL_N);
```

Входные параметры:

VMHL_Vector — указатель на вектор (одномерный массив);

VMHL_N — количество элементов в массиве.

Возвращаемое значение: Максимальный элемент.

Код 69. Пример использования

```
int VMHL_N=10;//Размер массива
double max;
double *a;
a=new double[VMHL_N];

for (int i=0;i<VMHL_N;i++) a[i]=MHL_RandomNumber();//Заполняем случайными зна
    чениями

//Вызов функции
max=TMHL_MaximumOfVector(a,VMHL_N);

//Используем полученный результат
MHL_ShowVector (a,VMHL_N,"Заполненный вектор", "a");
//Заполненный вектор:
//a =
//0.0988159
//0.61557
//0.674866
//0.937286
//0.521759
//0.074585
//0.733337
//0.5979
//0.604309
//0.917114

MHL_ShowNumber (max,"Максимальное значение в векторе", "max");
//Максимальное значение в векторе:
//max=0.937286

delete [] a;
```

6.1.9 TMHL_MinimumOfVector

Функция ищет минимальный элемент в векторе (одномерном массиве).

Код 70. Синтаксис

```
template <class T> T TMHL_MinimumOfVector(T *VMHL_Vector, int VMHL_N);
```

Входные параметры:

VMHL_Vector — указатель на вектор (одномерный массив);

VMHL_N — количество элементов в массиве.

Возвращаемое значение: Минимальный элемент.

Код 71. Пример использования

```
int VMHL_N=10;//Размер массива
double min;
double *a;
a=new double[VMHL_N];

for (int i=0;i<VMHL_N;i++) a[i]=MHL_RandomNumber();//Заполняем случайными зна
    чениями
```

```
        //Вызов функции
        min=TMHL_MinimumOfVector(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Заполненный вектор", "a");
        //Заполненный вектор:
        //a =
        //0.777496
        //0.446411
        //0.14621
        //0.938232
        //0.354156
        //0.831604
        //0.420349
        //0.50061
        //0.491394
        //0.0112305

        MHL_ShowNumber (min,"Минимальное значение в векторе", "min");
        //Максимальное значение в векторе:
        //max=0.0112305

        delete [] a;
```

6.1.10 TMHL_MixingVector

Функция перемешивает массив. Поочередно рассматриваются номера элементов массивов. С некоторой вероятностью рассматриваемый элемент массива меняется местами со случайным элементом массива.

Код 72. Синтаксис

```
template <class T> void TMHL_MixingVector(T *VMHL_ResultVector, double P, int VMHL_N)
    ;
```

Входные параметры:

VMHL_ResultVector — указатель на исходный массив;

P — вероятность того, что элемент массива под рассматриваемым номером поменяется местами с каким—нибудь другим элементов (не включая самого себя);

VMHL_N — размер массива.

Возвращаемое значение:

Отсутствует.

Код 73. Пример использования

```
        int VMHL_N=10;//Размер массива
        int *x;
        x=new int[VMHL_N];
        //Заполним массив номерами от 1
        TMHL_OrdinalVector(x,VMHL_N);
        MHL_ShowVector (x,VMHL_N,"Вектор", "x");
```

```
        //Вектор:
        //x =
        //1
        //2
        //3
        //4
        //5
        //6
        //7
        //8
        //9
        //10

        double P=0.4;//Вероятность перемешивания

        //Вызов функции
        TMHL_MixingVector(x,P,VMHL_N);//Перемешаем массив

        //Используем полученный результат
        MHL_ShowVector (x,VMHL_N,"Перемешанный вектор", "x");
        //Перемешанный вектор:
        //x =
        //4
        //2
        //1
        //3
        //5
        //6
        //7
        //8
        //9
        //10

        delete [] x;
```

6.1.11 TMHL_MixingVectorWithConjugateVector

Функция перемешивает массив вместе со сопряженным массивом. Поочередно рассматриваются номера элементов массивов. С некоторой вероятностью рассматриваемый элемент массива меняется местами со случайным элементом массива. Пары элементов первого массива и сопряженного остаются без изменения.

Код 74. Синтаксис

```
template <class T, class T2> void TMHL_MixingVectorWithConjugateVector(T *
    VMHL_ResultVector, T2 *VMHL_ResultVector2, double P, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на исходный массив;

VMHL_ResultVector2 — указатель на сопряженный массив;

P — вероятность того, что элемент массива под рассматриваемым номером поменяется местами с каким—нибудь другим элементов (не включая самого себя);

VMHL_N — количество элементов в массивах.

Возвращаемое значение:

Отсутствует.

Код 75. Пример использования

```
int VMHL_N=10;//Размер массива
int *x;
x=new int[VMHL_N];
int *y;
y=new int[VMHL_N];
//Заполним массив номерами от 1
TMHL_OrdinalVector(x,VMHL_N);
//А сопряженный заполним номерами с нуля
TMHL_OrdinalVectorZero(y,VMHL_N);
MHL_ShowVectorT (x,VMHL_N,"Вектор", "x");
//Вектор:
//x =
//1 2  3  4  5  6  7  8  9  10

MHL_ShowVectorT (y,VMHL_N,"Вектор", "y");
//Вектор:
//y =
//0 1  2  3  4  5  6  7  8  9

double P=0.4;//Вероятность перемешивания

//Вызов функции
TMHL_MixingVectorWithConjugateVector(x,y,P,VMHL_N);//Перемешаем массив

//Используем полученный результат
MHL_ShowVectorT (x,VMHL_N,"Перемешанный вектор", "x");
// Перемешанный вектор:
//x =
//9 1  4  8  10 5  7  3  6  2

MHL_ShowVectorT (y,VMHL_N,"Сопряженный перемешанный вектор", "y");
//Сопряженный перемешанный вектор:
//y =
//8 0  3  7  9  4  6  2  5  1

delete [] x;
delete [] y;
```

6.1.12 TMHL_NumberOfDifferentValuesInVector

Функция подсчитывает число различных значений в векторе (одномерном массиве).

Код 76. Синтаксис

```
template <class T> int TMHL_NumberOfDifferentValuesInVector(T *a, int VMHL_N);
```

Входные параметры:

a — указатель на вектор;

VMHL_N — размер массива a.

Возвращаемое значение:

Отсутствует.

Примечание: Алгоритм очень топорный и медленный.

Код 77. Пример использования

```
        int i;
        int VMHL_N=10;//Размер массива (число строк)
        int *a;
        a=new int[VMHL_N];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniformInt(0,5);

        //Вызов функции
        int NumberOfDifferent=TMHL_NumberOfDifferentValuesInVector(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Случайный вектор", "a");
        //Случайный вектор:
        //a =
        //2
        //1
        //1
        //4
        //0
        //2
        //1
        //1
        //2
        //2

        MHL_ShowNumber (NumberOfDifferent,"Число различных значений в векторе", "
            NumberOfDifferent");
        //Число различных значений в векторе:
        //NumberOfDifferent=4
        delete [] a;
```

6.1.13 TMHL_NumberOfMaximumOfVector

Функция ищет номер максимального элемента в векторе (одномерном массиве).

Код 78. Синтаксис

```
template <class T> int TMHL_NumberOfMaximumOfVector(T *a, int VMHL_N);
```

Входные параметры:

a — указатель на вектор (одномерный массив);

VMHL_N — размер массива.

Возвращаемое значение:

Номер максимального элемента.

Код 79. Пример использования

```
        int i;
```

```
int VMHL_N=10;//Размер массива
double *Vector=new double[VMHL_N];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++) Vector[i]=MHL_RandomNumber();

//Вызов функции
double Number=TMHL_NumberOfMaximumOfVector(Vector,VMHL_N);

//Используем полученный результат
MHL_ShowVector (Vector,VMHL_N,"Случайный массив", "Vector");
//Случайный массив:
//Vector =
//0.9245
//0.221466
//0.301544
//0.643951
//0.881958
//0.832764
//0.104462
//0.0611267
//0.943604
//0.335205

MHL_ShowNumber(Number,"Номер максимального элемента","Number");//Например, вы
    водим результат
// Номер максимального элемента:
//Number=8
delete [] Vector;
```

6.1.14 TMHL_NumberOfMinimumOfVector

Функция ищет номер минимального элемента в векторе (одномерном массиве).

Код 80. Синтаксис

```
template <class T> int TMHL_NumberOfMinimumOfVector(T *a, int VMHL_N);
```

Входные параметры:

a — указатель на вектор (одномерный массив);

VMHL_N — размер массива.

Возвращаемое значение:

Номер минимального элемента.

Код 81. Пример использования

```
int i;
int VMHL_N=10;//Размер массива
double *Vector=new double[VMHL_N];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++) Vector[i]=MHL_RandomNumber();

//Вызов функции
double Number=TMHL_NumberOfMinimumOfVector(Vector,VMHL_N);

//Используем полученный результат
```

```
MHL_ShowVector (Vector,VMHL_N,"Случайный массив", "Vector");
//Случайный массив:
//Vector =
//0.958344
//0.0968323
//0.689697
//0.102264
//0.142242
//0.135925
//0.473816
//0.0245056
//0.616333
//0.798065

MHL_ShowNumber(Number,"Номер минимального элемента","Number");//Например, выв
    одим результат
//Номер минимального элемента:
//Number=7
delete [] Vector;
```

6.1.15 TMHL_NumberOfNegativeValues

Функция подсчитывает число отрицательных значений в векторе (одномерном массиве).

Код 82. Синтаксис

```
template <class T> int TMHL_NumberOfNegativeValues(T *a, int VMHL_N);
```

Входные параметры:

a — указатель на вектор (одномерный массив);

VMHL_N — размер массива.

Возвращаемое значение:

Число отрицательных значений в массиве.

Код 83. Пример использования

```
int i;
int VMHL_N=10;//Размер массива (число строк)
int *a;
a=new int[VMHL_N];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 a[i]=MHL_RandomUniformInt(-20,20);

//Вызов функции
int NumberOfNegative=TMHL_NumberOfNegativeValues(a,VMHL_N);

//Используем полученный результат
MHL_ShowVector (a,VMHL_N,"Случайный вектор", "a");
//Случайный вектор:
//a =
//12
//19
//-11
//-20
```

```
        //13
        //4
        //-6
        //-1
        //1
        //-8

        MHL_ShowNumber (NumberOfNegative,"Число отрицательных значений в векторе", "
            NumberOfNegative");
        //Число отрицательных значений в векторе:
        //NumberOfNegative=5

        delete [] a;
```

6.1.16 TMHL_NumberOfPositiveValues

Функция подсчитывает число положительных значений в векторе (одномерном массиве).

Код 84. Синтаксис

```
template <class T> int TMHL_NumberOfPositiveValues(T *a, int VMHL_N);
```

Входные параметры:

a — указатель на вектор (одномерный массив);

VMHL_N — размер массива.

Возвращаемое значение:

Число положительных значений в массиве.

Код 85. Пример использования

```
        int i;
        int VMHL_N=10;//Размер массива (число строк)
        int *a;
        a=new int[VMHL_N];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniformInt(-20,20);

        //Вызов функции
        int NumberOfNegative=TMHL_NumberOfPositiveValues(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Случайный вектор", "a");
        //Случайный вектор:
        //a =
        //6
        //14
        //14
        //13
        //-19
        //-18
        //11
        //-18
        //-20
        //5
```

```
        MHL_ShowNumber (NumberOfNegative,"Число положительных значений в векторе", "
            NumberOfNegative");
        //Число положительных значений в векторе:
        //NumberOfNegative=6

        delete [] a;
```

6.1.17 TMHL_NumberOfZeroValues

Функция подсчитывает число нулевых значений в векторе (одномерном массиве).

Код 86. Синтаксис

```
template <class T> int TMHL_NumberOfZeroValues(T *a, int VMHL_N);
```

Входные параметры:

a — указатель на вектор (одномерный массив);

VMHL_N — размер массива.

Возвращаемое значение:

Число нулевых значений в массиве.

Код 87. Пример использования

```
        int i;
        int VMHL_N=10;//Размер массива (число строк)
        int *a;
        a=new int[VMHL_N];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniformInt(-2,2);

        //Вызов функции
        int NumberOfNegative=TMHL_NumberOfZeroValues(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Случайный вектор", "a");
        //Случайный вектор:
        //a =
        //1
        //1
        //0
        //0
        //0
        //-1
        //1
        //1
        //0
        //1

        MHL_ShowNumber (NumberOfNegative,"Число нулевых значений в векторе", "
            NumberOfNegative");
        //Число нулевых значений в векторе:
        //NumberOfNegative=4
```

```
delete [] a;
```

6.1.18 TMHL_OrdinalVector

Функция заполняет вектор значениями, равные номеру элемента, начиная с единицы.

Код 88. Синтаксис

```
template <class T> void TMHL_OrdinalVector(T *VMHL_ResultVector, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на вектор (одномерный массив), который и заполняется;

VMHL_N — размер массива.

Возвращаемое значение:

Отсутствует.

Код 89. Пример использования

```
int VMHL_N=10;//Размер массива (число строк)
double *a;
a=new double[VMHL_N];

//Вызов функции
TMHL_OrdinalVector(a,VMHL_N);
//Вектор:
//a =
//1
//2
//3
//4
//5
//6
//7
//8
//9
//10

//Используем полученный результат
MHL_ShowVector (a,VMHL_N,"Вектор", "a");

delete [] a;
```

6.1.19 TMHL_OrdinalVectorZero

Функция заполняет вектор значениями, равные номеру элемента, начиная с нуля.

Код 90. Синтаксис

```
template <class T> void TMHL_OrdinalVectorZero(T *VMHL_ResultVector, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на вектор (одномерный массив), который и заполняется;

VMHL_N — размер массива.

Возвращаемое значение:

Отсутствует.

Код 91. Пример использования

```
int VMHL_N=10;//Размер массива (число строк)
double *a;
a=new double[VMHL_N];

//Вызов функции
TMHL_OrdinalVectorZero(a,VMHL_N);
//Вектор:
//a =
//0
//1
//2
//3
//4
//5
//6
//7
//8
//9

//Используем полученный результат
MHL_ShowVector (a,VMHL_N,"Вектор", "a");

delete [] a;
```

6.1.20 TMHL_ReverseVector

Функция меняет порядок элементов в массиве на обратный. Преобразуется подаваемый массив.

Код 92. Синтаксис

```
template <class T> void TMHL_ReverseVector(T *VMHL_ResultVector, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на преобразуемый массив;

VMHL_N — размер массива.

Возвращаемое значение:

Отсутствует.

Код 93. Пример использования

```
int i;
int VMHL_N=MHL_RandomUniformInt(2,10);//Размер массива (число строк)
double *a;
a=new double[VMHL_N];
```

```
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniformInt(10,100);

        MHL_ShowVector (a,VMHL_N,"Вектор равен", "a");
        //Вектор равен:
        //a =
        //83
        //57
        //55
        //52
        //70
        //73

        //Вызов функции
        TMHL_ReverseVector(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Теперь вектор равен", "a");
        //Теперь вектор равен:
        //a =
        //73
        //70
        //52
        //55
        //57
        //83

        delete [] a;
```

6.1.21 TMHL_SearchFirstNotZero

Функция возвращает номер первого ненулевого элемента массива.

Код 94. Синтаксис

```
template <class T> int TMHL_SearchFirstNotZero(T *x, int VMHL_N);
```

Входные параметры:

x — указатель на вектор (одномерный массив);

VMHL_N — размер массива.

Возвращаемое значение:

Номер первого ненулевого элемента массива (начиная с нуля). Если такого элемента нет, то возвращается -1.

Код 95. Пример использования

```
        int i;
        int VMHL_N=10;//Размер массива (число строк)
        int *a;
        a=new int[VMHL_N];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniformInt(0,2);
```

```
        //Вызов функции
        int Number=TMHL_SearchFirstNotZero(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Случайный вектор", "a");
        //Случайный вектор:
        //a =
        //0
        //0
        //0
        //1
        //0
        //0
        //0
        //1
        //1
        //0

        MHL_ShowNumber (Number,"Номер первого ненулевого элемента", "Number");
        //Номер первого ненулевого элемента:
        //Number=3

        delete [] a;
```

6.1.22 TMHL_SearchFirstZero

Функция возвращает номер первого нулевого элемента массива.

Код 96. Синтаксис

```
template <class T> int TMHL_SearchFirstZero(T *x, int VMHL_N);
```

Входные параметры:

x — указатель на вектор (одномерный массив);

VMHL_N — размер массива.

Возвращаемое значение:

Номер первого нулевого элемента массива (начиная с нуля). Если такого элемента нет, то возвращается -1.

Код 97. Пример использования

```
        int i;
        int VMHL_N=10;//Размер массива (число строк)
        int *a;
        a=new int[VMHL_N];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniformInt(0,2);

        //Вызов функции
        int Number=TMHL_SearchFirstZero(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Случайный вектор", "a");
        //Случайный вектор:
```

```
//a =
//1
//1
//1
//0
//0
//1
//0
//0
//0
//1

MHL_ShowNumber (Number,"Номер первого нулевого элемента", "Number");
//Номер первого нулевого элемента:
//Number=3

delete [] a;
```

6.1.23 TMHL_SumSquareVector

Функция вычисляет сумму квадратов элементов вектора.

Код 98. Синтаксис

```
template <class T> T TMHL_SumSquareVector(T *VMHL_Vector,int VMHL_N);
```

Входные параметры:

VMHL_Vector — указатель на вектор (одномерный массив);

VMHL_N — количество элементов в массиве.

Возвращаемое значение: Сумма квадратов элементов массива.

Код 99. Пример использования

```
int VMHL_N=10;//Размер массива
double sum;
double *a;
a=new double[VMHL_N];

for (int i=0;i<VMHL_N;i++) a[i]=i;//Заполняем значениями

//Вызов функции
sum=TMHL_SumSquareVector(a,VMHL_N);

//Используем полученный результат
MHL_ShowVector (a,VMHL_N,"Заполненный вектор", "a");
//Заполненный вектор:
//a =
//0
//1
//2
//3
//4
//5
//6
//7
//8
```

```
        //9

        MHL_ShowNumber (sum,"Сумма квадратов элементов массива", "sum");
        //Сумма квадратов элементов массива:
        //sum=285

        delete [] a;
```

6.1.24 TMHL_SumVector

Функция вычисляет сумму элементов вектора.

Код 100. Синтаксис

```
template <class T> T TMHL_SumVector(T *VMHL_Vector,int VMHL_N);
```

Входные параметры:

VMHL_Vector — указатель на вектор (одномерный массив);

VMHL_N — количество элементов в массиве.

Возвращаемое значение: Сумма элементов вектора.

Код 101. Пример использования

```
        int VMHL_N=10;//Размер массива
        double sum;
        double *a;
        a=new double[VMHL_N];

        for (int i=0;i<VMHL_N;i++) a[i]=MHL_RandomNumber();//Заполняем случайными зна
            чениями

        //Вызов функции
        sum=TMHL_SumVector(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Заполненный вектор", "a");
        //Заполненный вектор:
        //a =
        //0.886475
        //0.998413
        //0.242859
        //0.221405
        //0.292175
        //0.134247
        //0.723846
        //0.271393
        //0.188904
        //0.727936

        MHL_ShowNumber (sum,"Сумма элементов массива", "sum");
        //Сумма элементов массива:
        //sum=4.68765

        delete [] a;
```

6.1.25 TMHL_VectorMinusVector

Функция вычитает поэлементно из одного массива другой и записывает результат в третий массив. Или в переопределенном виде функция вычитает поэлементно из одного массива другой и записывает результат в первый массив.

Код 102. Синтаксис

```
template <class T> void TMHL_VectorMinusVector(T *a, T *b, T *VMHL_ResultVector, int
    VMHL_N);
template <class T> void TMHL_VectorMinusVector(T *VMHL_ResultVector, T *b, int VMHL_N
    );
```

Входные параметры:

a — первый вектор;

b — вторый вектор;

VMHL_ResultVector — вектор разности;

VMHL_N — размер векторов.

Возвращаемое значение:

Отсутствует.

Для переопределенной функции:

Входные параметры:

VMHL_ResultVector — первый вектор, из которого вычитают второй вектор;

b — вторый вектор;

VMHL_N — размер векторов.

Возвращаемое значение:

Отсутствует.

Код 103. Пример использования

```
        int i;
        int VMHL_N=10;//Размер массива (число строк)
        int *a;
        a=new int[VMHL_N];
        int *b;
        b=new int[VMHL_N];
        int *c;
        c=new int[VMHL_N];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniformInt(0,10);
        for (i=0;i<VMHL_N;i++)
         b[i]=MHL_RandomUniformInt(0,10);

        //Вызов функции
        TMHL_VectorMinusVector(a,b,c,VMHL_N);

        //Используем полученный результат
```

```
  MHL_ShowVectorT (a,VMHL_N,"Случайный вектор", "a");
  //Случайный вектор:
  //a =
  //0 7  0  0  8  5  0  4  8  2

  MHL_ShowVectorT (b,VMHL_N,"Случайный вектор", "b");
  //Случайный вектор:
  //b =
  //6 1  3  1  2  7  2  6  1  4

  MHL_ShowVectorT (c,VMHL_N,"Их разница", "c");
  //Их разница:
  //c =
  //-6   6  -3 -1 6  -2 -2 -2 7  -2

  delete [] a;
  delete [] b;
  delete [] c;



//Для  переопределенной функции
  VMHL_N=10;//Размер массива (число строк)
  a=new int[VMHL_N];
  b=new int[VMHL_N];
  //Заполним случайными числами
  for (i=0;i<VMHL_N;i++)
   a[i]=MHL_RandomUniformInt(0,10);
  for (i=0;i<VMHL_N;i++)
   b[i]=MHL_RandomUniformInt(0,10);

  MHL_ShowVectorT (a,VMHL_N,"Случайный вектор", "a");
  //Случайный вектор:
  //a =
  //6 9  3  0  2  9  4  2  3  7

  //Вызов функции
  TMHL_VectorMinusVector(a,b,VMHL_N);

  //Используем полученный результат
  MHL_ShowVectorT (b,VMHL_N,"Случайный вектор", "b");
  //Случайный вектор:
  //b =
  //5 6  3  8  5  0  7  6  4  4

  MHL_ShowVectorT (a,VMHL_N,"Из первого вычли второй", "a");
  //Из первого вычли второй:
  //a =
  //1 3  0  -8 -3 9  -3 -4 -1 3

  delete [] a;
  delete [] b;
```

6.1.26 TMHL_VectorMultiplyNumber

Функция умножает вектор на число.

Код 104. Синтаксис

```
template <class T> void TMHL_VectorMultiplyNumber(T *VMHL_ResultVector, int VMHL_N, T
    Number);
```

Входные параметры:

VMHL_ResultVector — вектор (в нем и сохраняется результат);

VMHL_N — размер вектора;

Number — число, на которое умножается вектор.

Возвращаемое значение:

Отсутствует.

Код 105. Пример использования

```
        int i;
        int VMHL_N=10;//Размер массива (число строк)
        double *a;
        a=new double[VMHL_N];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniformInt(0,10);

        MHL_ShowVector (a,VMHL_N,"Случайный вектор", "a");
        //Случайный вектор:
        //a =
        //4
        //6
        //3
        //5
        //4
        //7
        //8
        //2
        //1
        //0

        double Number=MHL_RandomUniform(0,10);

        //Вызов функции
        TMHL_VectorMultiplyNumber(a,VMHL_N,Number);

        //Используем полученный результат
        MHL_ShowNumber (Number,"Случайный множитель", "Number");
        //Случайный множитель:
        //Number=3.57941

        MHL_ShowVector (a,VMHL_N,"Умножили на число Number", "a");
        //Умножили на число Number:
        //a =
        //14.3176
        //21.4764
        //10.7382
        //17.897
        //14.3176
        //25.0558
        //28.6353
        //7.15881
        //3.57941
```

```
        //0

        delete [] a;
```

6.1.27 TMHL_VectorPlusVector

Функция складывает поэлементно из одного массива другой и записывает результат в третий массив. Или в переопределенном виде функция складывает поэлементно из одного массива другой и записывает результат в первый массив.

Код 106. Синтаксис

```
template <class T> void TMHL_VectorPlusVector(T *a, T *b, T *VMHL_ResultVector, int
    VMHL_N);
template <class T> void TMHL_VectorPlusVector(T *VMHL_ResultVector, T *b, int VMHL_N)
    ;
```

Входные параметры:

a — первый вектор;

b — вторый вектор;

VMHL_ResultVector — вектор суммы;

VMHL_N — размер векторов.

Возвращаемое значение:

Отсутствует.

Для переопределенной функции:

Входные параметры:

VMHL_ResultVector — первый вектор, к которому прибавляют второй вектор;

b — вторый вектор;

VMHL_N — размер векторов.

Возвращаемое значение:

Отсутствует.

Код 107. Пример использования

```
        int i;
        int VMHL_N=10;//Размер массива (число строк)
        int *a;
        a=new int[VMHL_N];
        int *b;
        b=new int[VMHL_N];
        int *c;
        c=new int[VMHL_N];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniformInt(0,10);
        for (i=0;i<VMHL_N;i++)
```

```
 b[i]=MHL_RandomUniformInt(0,10);

//Вызов функции
TMHL_VectorPlusVector(a,b,c,VMHL_N);

//Используем полученный результат
MHL_ShowVectorT (a,VMHL_N,"Случайный вектор", "a");
//Случайный вектор:
//a =
//2 7  9  2  3  3  3  2  8  8


MHL_ShowVectorT (b,VMHL_N,"Случайный вектор", "b");
//Случайный вектор:
//b =
//3 7  2  9  5  3  2  7  2  7

MHL_ShowVectorT (c,VMHL_N,"Их сумма", "c");
//Их сумма:
//c =
//5 14 11 11 8  6  5  9  10 15

delete [] a;
delete [] b;
delete [] c;

//Для  переопределенной функции
VMHL_N=10;//Размер массива (число строк)
a=new int[VMHL_N];
b=new int[VMHL_N];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 a[i]=MHL_RandomUniformInt(0,10);
for (i=0;i<VMHL_N;i++)
 b[i]=MHL_RandomUniformInt(0,10);

MHL_ShowVectorT (a,VMHL_N,"Случайный вектор", "a");
//Случайный вектор:
//a =
//0 6  7  4  9  3  9  8  5  6

//Вызов функции
TMHL_VectorPlusVector(a,b,VMHL_N);

//Используем полученный результат
MHL_ShowVectorT (b,VMHL_N,"Случайный вектор", "b");
//Случайный вектор:
//b =
//1 7  0  5  4  0  9  5  7  7

MHL_ShowVectorT (a,VMHL_N,"К первому прибавили второй", "a");
//К первому прибавили второй:
//a =
//1 13 7  9  13 3  18 13 12 13

delete [] a;
delete [] b;
```

6.1.28 TMHL_VectorToVector

Функция копирует содержимое вектора (одномерного массива) в другой.

Код 108. Синтаксис

```
template <class T> void TMHL_VectorToVector(T *VMHL_Vector, T *VMHL_ResultVector, int
    VMHL_N);
```

Входные параметры:

VMHL_Vector — указатель на исходный массив;

VMHL_ResultVector — указатель на массив в который производится запись;

VMHL_N — размер массивов.

Возвращаемое значение: Отсутствует.

Код 109. Пример использования

```
        int VMHL_N=10;//Размер массива

        double *a;
        a=new double[VMHL_N];
        for (int i=0;i<VMHL_N;i++) a[i]=MHL_RandomNumber();//Заполняем случайными зна
            чениями

        double *b;
        b=new double[VMHL_N];

        //Вызов функции
        TMHL_VectorToVector(a,b,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Первоначальный вектор", "a");
        //Первоначальный вектор:
        //a =
        //0.874634
        //0.28656
        //0.676056
        //0.861755
        //0.0521851
        //0.308319
        //0.348267
        //0.431671
        //0.186462
        //0.562805

        MHL_ShowVector (b,VMHL_N,"Вектор, в который скопировали первый", "b");
        //Вектор, в который скопировали первый:
        //b =
        //0.874634
        //0.28656
        //0.676056
        //0.861755
        //0.0521851
        //0.308319
        //0.348267
        //0.431671
        //0.186462
```

```
        //0.562805

        delete [] a;
        delete [] b;
```

6.1.29 TMHL_ZeroVector

Функция зануляет массив.

Код 110. Синтаксис

```
template <class T> void TMHL_ZeroVector(T *VMHL_ResultVector,int VMHL_N);
```

Входные параметры:

VMHL_Vector — указатель на вектор (одномерный массив);

VMHL_N — количество элементов в массиве.

Возвращаемое значение: Отсутствует.

Код 111. Пример использования

```
        int VMHL_N=10;//Размер массива
        double *a;
        a=new double[VMHL_N];

        //Вызов функции
        TMHL_ZeroVector(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Зануленный вектор", "a");
        //Зануленный вектор:
        //a =
        //0
        //0
        //0
        //0
        //0
        //0
        //0
        //0
        //0
        //0

        delete [] a;
```

6.2 Геометрия

6.2.1 TMHL_BoolCrossingTwoSegment

Функция определяет наличие пересечения двух отрезков. Координаты отрезков могут быть перепутаны по порядку в каждом отрезке.

Код 112. Синтаксис

```
template <class T> int TMHL_BoolCrossingTwoSegment(T b1,T c1,T b2,T c2);
```

Входные параметры:

b1 — левый конец первого отрезка;

c1 — правый конец первого отрезка;

b2 — левый конец второго отрезка;

c2 — правый конец второго отрезка.

Возвращаемое значение:

1 — отрезки пересекаются;

0 — отрезки не пересекаются.

Код 113. Пример использования

```
double b1,c1,b2,c2;
int Result;
//Зададим случайные координаты отрезков
b1=MHL_RandomUniform(-3,5);
c1=MHL_RandomUniform(-3,5);
b2=MHL_RandomUniform(-3,5);
c2=MHL_RandomUniform(-3,5);

//Вызов функции
Result=TMHL_BoolCrossingTwoSegment(b1,c1,b2,c2);

//Используем полученный результат
MHL_ShowNumber (b1,"Левый конец первого отрезка", "b1");
//Левый конец первого отрезка:
//b1=0.773193
MHL_ShowNumber (c1,"Правый конец первого отрезка", "c1");
//Правый конец первого отрезка:
//c1=3.22803
MHL_ShowNumber (b2,"Левый конец второго отрезка", "b2");
//Левый конец второго отрезка:
//b2=4.99121
MHL_ShowNumber (c2,"Правый конец второго отрезка", "c2");
//Правый конец второго отрезка:
//c2=1.43921
MHL_ShowNumber (Result,"Пересекаются ли отрезки", "Result");
//Пересекаются ли отрезки:
//Result=1
```

6.3 Гиперболические функции

6.3.1 MHL_Cosech

Функция возвращает гиперболический косеканс.

Код 114. Синтаксис

```
double MHL_Cosech(double x);
```

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический косеканс.

Код 115. Пример использования

```
double x=MHL_RandomUniform(0,10);

//Вызов функции
double Result=MHL_Cosech(x);

//Используем полученный результат
MHL_ShowNumber(Result,"Гиперболический косеканс от x="+MHL_NumberToText(x),"р
    авен");
//Гиперболический косеканс от x=0.571289:
//равен=1.65872
```

6.3.2 MHL_Cosh

Функция возвращает гиперболический косинус.

Код 116. Синтаксис

```
double MHL_Cosh(double x);
```

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический косинус.

Код 117. Пример использования

```
double x=MHL_RandomUniform(0,10);

//Вызов функции
double Result=MHL_Cosh(x);

//Используем полученный результат
MHL_ShowNumber(Result,"Гиперболический косинус от x="+MHL_NumberToText(x),"ра
    вен");
//Гиперболический косинус от x=4.04968:
//равен=28.6983
```

6.3.3 MHL_Cotanh

Функция возвращает гиперболический котангенс.

Код 118. Синтаксис

```
double MHL_Cotanh(double x);
```

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический котангенс.

Код 119. Пример использования

```
double x=MHL_RandomUniform(0,10);

//Вызов функции
double Result=MHL_Cotanh(x);

//Используем полученный результат
MHL_ShowNumber(Result,"Гиперболический котангенс от x="+MHL_NumberToText(x),"
    равен");
//Гиперболический котангенс от x=1.92505:
//равен=1.04348
```

6.3.4 MHL_Sech

Функция возвращает гиперболический секанс.

Код 120. Синтаксис

```
double MHL_Sech(double x);
```

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический секанс.

Код 121. Пример использования

```
double x=MHL_RandomUniform(0,10);

//Вызов функции
double Result=MHL_Sech(x);

//Используем полученный результат
MHL_ShowNumber(Result,"Гиперболический секанс от x="+MHL_NumberToText(x),"рав
    ен");
//Гиперболический секанс от x=0.679932:
//равен=0.806323
```

6.3.5 MHL_Sinh

Функция возвращает гиперболический синус.

Код 122. Синтаксис

```
double MHL_Sinh(double x);
```

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический синус.

Код 123. Пример использования

```
double x=MHL_RandomUniform(0,10);

//Вызов функции
double Result=MHL_Sinh(x);

//Используем полученный результат
MHL_ShowNumber(Result,"Гиперболический синус от x="+MHL_NumberToText(x),"раве
    н");
//Гиперболический синус от x=0.166321:
//равен=0.167089
```

6.3.6 MHL_Tanh

Функция возвращает гиперболический тангенс.

Код 124. Синтаксис

```
double MHL_Tanh(double x);
```

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический тангенс.

Код 125. Пример использования

```
double x=MHL_RandomUniform(0,10);

//Вызов функции
double Result=MHL_Tanh(x);

//Используем полученный результат
MHL_ShowNumber(Result,"Гиперболический тангенс от x="+MHL_NumberToText(x),"ра
    вен");
//Гиперболический тангенс от x=4.27643:
//равен=0.999614
```

6.4 Дифференцирование

6.4.1 MHL_CenterDerivative

Численное значение производной в точке (центральной разностной производной с шагом 2h).

Код 126. Синтаксис

```
double MHL_CenterDerivative(double x, double h, double (*Function)(double));
```

Входные параметры: x — точка, в которой считается производная;

h — малое приращение x;

Function — функция, производная которой ищется.

Возвращаемое значение:

Значение производной в точке.

Примечание:

При $h \leq 0$ возвращается 0.

Формула:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2 \cdot h}$$

Будем использовать в примере использования дополнительную функцию.

Код 127. Дополнительная функция

```
double Func3(double x)
{
return x*x;
}
//---------------------------------------------------------------------------
```

Код 128. Пример использования

```
        double x;
        double h;
        double dfdx;
        //Зададим случайные координаты отрезков
        x=MHL_RandomUniform(-3,5);
        h=0.01;//малое приращение x

        //Вызов функции
        dfdx=MHL_CenterDerivative(x,h,Func3);

        //Используем полученный результат
        MHL_ShowNumber (x,"Точка, в которой считается производная", "x");
        //Точка, в которой считается производная:
        //x=0.843262
        MHL_ShowNumber (h,"Малое приращение x", "h");
        // Малое приращение x:
        //h=0.01
        MHL_ShowNumber (dfdx,"Значение производной в точке", "dfdx");
        // Значение производной в точке:
        //dfdx=1.68652
```

6.4.2 MHL_LeftDerivative

Численное значение производной в точке (разностная производная влево).

Код 129. Синтаксис

```
double MHL_LeftDerivative(double x, double h, double (*Function)(double));
```

Входные параметры: x — точка, в которой считается производная;

h — малое приращение x;

Function — функция, производная которой ищется.

Возвращаемое значение:

Значение производной в точке.

Примечание:

При $h \leq 0$ возвращается 0.

Формула:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

Будем использовать в примере использования дополнительную функцию.

Код 130. Дополнительная функция

```
double Func3(double x)
{
return x*x;
}
//---------------------------------------------------------------------------
```

Код 131. Пример использования

```
        double x;
        double h;
        double dfdx;
        //Зададим случайные координаты отрезков
        x=MHL_RandomUniform(-3,5);
        h=0.01;//малое приращение x

        //Вызов функции
        dfdx=MHL_LeftDerivative(x,h,Func3);

        //Используем полученный результат
        MHL_ShowNumber (x,"Точка, в которой считается производная", "x");
        // Точка, в которой считается производная:
        //x=1.87964
        MHL_ShowNumber (h,"Малое приращение x", "h");
        // Малое приращение x:
        //h=0.01
        MHL_ShowNumber (dfdx,"Значение производной в точке", "dfdx");
        // Значение производной в точке:
        //dfdx=3.74928
```

6.4.3 MHL_RightDerivative

Численное значение производной в точке (разностная производная вправо).

Код 132. Синтаксис

```
double MHL_RightDerivative(double x, double h, double (*Function)(double));
```

Входные параметры: x — точка, в которой считается производная;

h — малое приращение x;

Function — функция, производная которой ищется.

Возвращаемое значение:

Значение производной в точке.

Примечание:

При $h \leq 0$ возвращается 0.

Формула:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Будем использовать в примере использования дополнительную функцию.

Код 133. Дополнительная функция

```
double Func3(double x)
{
return x*x;
}
//---------------------------------------------------------------------------
```

Код 134. Пример использования

```
        double x;
        double h;
        double dfdx;
        //Зададим случайные координаты отрезков
        x=MHL_RandomUniform(-3,5);
        h=0.01;//малое приращение x

        //Вызов функции
        dfdx=MHL_RightDerivative(x,h,Func3);

        //Используем полученный результат
        MHL_ShowNumber (x,"Точка, в которой считается производная", "x");
        // Точка, в которой считается производная:
        //x=-1.69409
        MHL_ShowNumber (h,"Малое приращение x", "h");
        // Малое приращение x:
        //h=0.01
        MHL_ShowNumber (dfdx,"Значение производной в точке", "dfdx");
        // Значение производной в точке:
        //dfdx=-3.37818
```

6.5 Интегрирование

6.5.1 MHL_IntegralOfRectangle

Интегрирование по формуле прямоугольников с оценкой точности по правилу Рунге. Считается интеграл функции на отрезке [a,b] с погрешностью порядка Epsilon.

Код 135. Синтаксис

```
double MHL_IntegralOfRectangle(double a, double b, double Epsilon,double (*Function)(
    double));
```

Входные параметры:

a — начало отрезка интегрирования;

b — конец отрезка интегрирования;

Epsilon — погрешность;

Function — подынтегральная функция.

Возвращаемое значение:

Значение определенного интеграла.

Примечание:

Значимые цифры в ответе определяются Epsilon.

Будем использовать в примере использования дополнительную подынтегральную функцию:

Код 136. Дополнительная функция

```
double Func3(double x)
{
return x*x;
}
//---------------------------------------------------------------------------
```

Код 137. Пример использования

```
        double a=-2;
        double b=2;
        double Epsilon=0.01;
        double S;

        //Вызов функции
        S=MHL_IntegralOfRectangle(a,b,Epsilon,Func3);

        //Используем полученный результат
        MHL_ShowNumber (a,"Левая граница интегрирования", "a");
        //Левая граница интегрирования:
        //a=-2
        MHL_ShowNumber (b,"Правая граница интегрирования", "b");
        //Правая граница интегрирования:
        //b=2
        MHL_ShowNumber (S,"Интеграл", "S");
        // Интеграл:
        //S=5.32812
```

6.5.2 MHL_IntegralOfSimpson

Интегрирование по формуле Симпсона с оценкой точности по правилу Рунге. Считается интеграл функции на отрезке [a,b] с погрешностью порядка Epsilon.

Код 138. Синтаксис

```
double MHL_IntegralOfSimpson(double a, double b, double Epsilon,double (*Function)(
    double));
```

Входные параметры:

a — начало отрезка интегрирования;

b — конец отрезка интегрирования;

Epsilon — погрешность;

Function — подынтегральная функция.

Возвращаемое значение:

Значение определенного интеграла.

Примечание:

Значимые цифры в ответе определяются Epsilon.

Будем использовать в примере использования дополнительную подынтегральную функцию:

Код 139. Дополнительная функция

```
double Func3(double x)
{
return x*x;
}
//---------------------------------------------------------------------------
```

Код 140. Пример использования

```
        double a=-2;
        double b=2;
        double Epsilon=0.01;
        double S;

        //Вызов функции
        S=MHL_IntegralOfSimpson(a,b,Epsilon,Func3);

        //Используем полученный результат
        MHL_ShowNumber (a,"Левая граница интегрирования", "a");
        // Левая граница интегрирования:
        //a=-2
        MHL_ShowNumber (b,"Правая граница интегрирования", "b");
        // Правая граница интегрирования:
        //b=2
        MHL_ShowNumber (S,"Интеграл", "S");
        // Интеграл:
        //S=5.33333
```

6.5.3 MHL_IntegralOfTrapezium

Интегрирование по формуле трапеции с оценкой точности по правилу Рунге. Считается интеграл функции на отрезке [a,b] с погрешностью порядка Epsilon.

Код 141. Синтаксис

```
double MHL_IntegralOfTrapezium(double a, double b, double Epsilon,double (*Function)(
    double));
```

Входные параметры:

a — начало отрезка интегрирования;

b — конец отрезка интегрирования;

Epsilon — погрешность;

Function — подынтегральная функция.

Возвращаемое значение:

Значение определенного интеграла.

Примечание:

Значимые цифры в ответе определяются Epsilon.

Будем использовать в примере использования дополнительную подынтегральную функцию:

Код 142. Дополнительная функция

```
double Func3(double x)
{
return x*x;
}
//---------------------------------------------------------------------------
```

Код 143. Пример использования

```
        double a=-2;
        double b=2;
        double Epsilon=0.01;
        double S;

        //Вызов функции
        S=MHL_IntegralOfTrapezium(a,b,Epsilon,Func3);

        //Используем полученный результат
        MHL_ShowNumber (a,"Левая граница интегрирования", "a");
        //Левая граница интегрирования:
        //a=-2
        MHL_ShowNumber (b,"Правая граница интегрирования", "b");
        //Правая граница интегрирования:
        //b=2
        MHL_ShowNumber (S,"Интеграл", "S");
        //Интеграл:
        //S=5.33594
```

6.6 Математические функции

6.6.1 MHL_ArithmeticalProgression

Арифметическая прогрессия. n-ый член последовательности.

Код 144. Синтаксис

```
double MHL_ArithmeticalProgression(double a1,double d,int n);
```

Входные параметры:

a1 — начальный член прогрессии;

d — шаг арифметической прогрессии;

n — номер последнего члена.

Возвращаемое значение:

n-ый член последовательности.

Код 145. Пример использования

```
        double a1=MHL_RandomUniformInt(1,10);
        double d=MHL_RandomUniformInt(1,10);
        int n=MHL_RandomUniformInt(1,10);

        double an=MHL_ArithmeticalProgression(a1,d,n);

        //Используем полученный результат
        MHL_ShowNumber(a1,"Первый член последовательности","a1");
        //Первый член последовательности:
        //a1=6
        MHL_ShowNumber(d,"Шаг арифметической прогрессии","d");
        //Шаг арифметической прогрессии:
        //d=9
        MHL_ShowNumber(n,"Номер последнего члена","n");
        //Номер последнего члена:
        //n=4
        MHL_ShowNumber(an,"n-ый член последовательности","an");
        //n-ый член последовательности:
        //an=33
```

6.6.2 MHL_ExpMSxD2

Функция вычисляет выражение $exp(-x * x/2)$.

Код 146. Синтаксис

```
double MHL_ExpMSxD2(double x);
```

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Значение функции в точке.

Формула:

$$F(x) = e^{-\frac{x^2}{2}}.$$

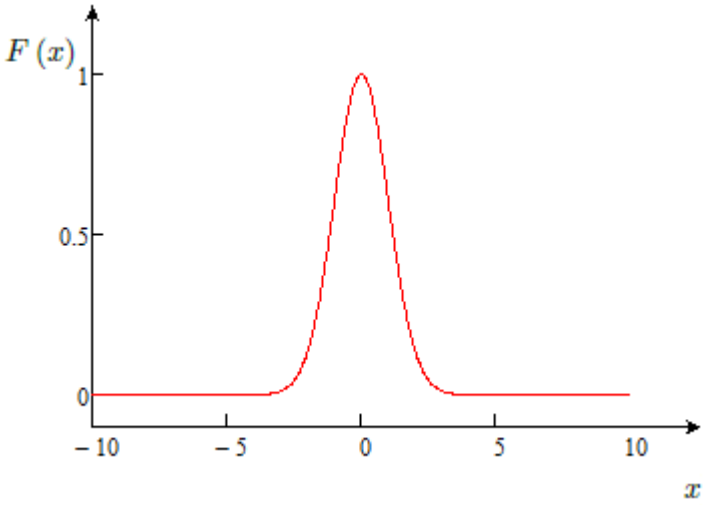

Рисунок 1. График функции

Код 147. Пример использования

```
double t;
double f;
t=MHL_RandomUniform(0,3);

//Вызов функции
f=MHL_ExpMSxD2(t);

//Используем полученный результат

MHL_ShowNumber (t,"Параметр", "t");
//Параметр:
//t=2.06177
MHL_ShowNumber (f,"Значение функции", "f");
//Значение функции:
//f=0.11938
```

6.6.3 MHL_GeometricSeries

Геометрическая прогрессия. n-ый член последовательности.

Код 148. Синтаксис

```
double MHL_GeometricSeries(double u1,double q,int n);
```

Входные параметры:

u1 — начальный член прогрессии;

q — шаг прогрессии;

n — номер последнего члена.

Возвращаемое значение:

n-ый член последовательности.

Код 149. Пример использования

```
        double u1=MHL_RandomUniformInt(1,10);
        double q=MHL_RandomUniformInt(1,10);
        int n=MHL_RandomUniformInt(1,10);

        double qn=MHL_GeometricSeries(u1,q,n);

        //Используем полученный результат
        MHL_ShowNumber(u1,"Первый член последовательности","u1");
        //Первый член последовательности:
        //u1=4
        MHL_ShowNumber(q,"Шаг арифметической прогрессии","q");
        //Шаг арифметической прогрессии:
        //q=4
        MHL_ShowNumber(n,"Номер последнего члена","n");
        //Номер последнего члена:
        //n=6
        MHL_ShowNumber(qn,"n-ый член последовательности","qn");
        //n-ый член последовательности:
        //qn=4096
```

6.6.4 MHL_GreatestCommonDivisorEuclid

Функция находит наибольший общий делитель двух чисел по алгоритму Евклида.

Код 150. Синтаксис

```
int MHL_GreatestCommonDivisorEuclid(int A,int B);
```

Входные параметры:

A — первое число;

B — второе число.

Возвращаемое значение:

НОД(A,B)

Код 151. Пример использования

```
        int A=MHL_RandomUniformInt(1,100);
        int B=MHL_RandomUniformInt(1,100);

        double Result=MHL_GreatestCommonDivisorEuclid(A,B);

        //Используем полученный результат
        MHL_ShowNumber(A,"Число","A");
        //Число:
        //A=96
        MHL_ShowNumber(B,"Число","B");
        //Число:
```

```
        //B=18
        MHL_ShowNumber(Result,"НОД","");
        //НОД:
        //=6
```

6.6.5 MHL_HowManyPowersOfTwo

Функция вычисляет, какой минимальной степенью двойки можно покрыть целое положительное число.

Код 152. Синтаксис

```
int MHL_HowManyPowersOfTwo(int x);
```

Входные параметры:

x — целое число.

Возвращаемое значение:

Минимальная степень двойки можно покрыть целое положительное число: $2^{VMHL\_Result} > x$

Код 153. Пример использования

```
        int x=MHL_RandomUniformInt(0,1000);
        int Degree;

        //Вызываем функцию
        Degree=MHL_HowManyPowersOfTwo(x);

        //Используем полученный результат
        MHL_ShowNumber(x,"Число","x");
        //Число:
        //x=480
        MHL_ShowNumber(Degree,"Его покрывает 2 в степени","Degree");
        //Его покрывает 2 в степени:
        //Degree=9
        MHL_ShowNumber(TMHL_PowerOf(2,Degree),"То есть","2^"+MHL_NumberToText(Degree)
            );
        //То есть:
        //2^9=512
```

6.6.6 MHL_InverseNormalizationNumberAll

Функция осуществляет обратную нормировку числа из интервала $[0;1]$ в интервал $[-\infty;\infty]$, которое было осуществлено функцией MHL_NormalizationNumberAll.

Код 154. Синтаксис

```
double MHL_InverseNormalizationNumberAll(double x);
```

Входные параметры:

x — число в интервале [0;1].

Возвращаемое значение:

Перенормированное число.

Код 155. Пример использования

```
        double x;
        double y;
        x=MHL_RandomNumber();

        //Вызов функции
        y=MHL_InverseNormalizationNumberAll(x);

        //Используем полученный результат
        MHL_ShowNumber (x,"Нормированное число", "x");
        // Нормированное число:
        //x=0.0491333
        MHL_ShowNumber (y,"Перенормированное число", "y");
        // Перенормированное число:
        //y=-9.1764
```

6.6.7 MHL_LeastCommonMultipleEuclid

Функция находит наименьшее общее кратное двух чисел по алгоритму Евклида.

Код 156. Синтаксис

```
int MHL_LeastCommonMultipleEuclid(int A,int B);
```

Входные параметры:

A — первое число;

B — второе число.

Возвращаемое значение:

НОК(A,B)

Код 157. Пример использования

```
        int A=MHL_RandomUniformInt(1,100);
        int B=MHL_RandomUniformInt(1,100);

        double Result=MHL_LeastCommonMultipleEuclid(A,B);

        //Используем полученный результат
        MHL_ShowNumber(A,"Число","A");
        //Число:
        //A=68
        MHL_ShowNumber(B,"Число","B");
        //Число:
        //B=67
        MHL_ShowNumber(Result,"НОК","");
        //НОК:
        //=4556
```

6.6.8 MHL_NormalizationNumberAll

Функция нормирует число из интервала $[-\infty;\infty]$ в интервал $[0;1]$. При этом в нуле возвращает 0.5, в $-\infty$ возвращает 0, в ∞ возвращает 1. Если $x < y$, то $MHL\_NormalizationNumberAll(x) < MHL\_NormalizationNumberAll(y)$. Под бесконечностью принимается машинная бесконечность.

Код 158. Синтаксис

```
double MHL_NormalizationNumberAll(double x);
```

Входные параметры:

x — число.

Возвращаемое значение:

Нормированное число.

Формула:

$$f(x) = \frac{1}{2}\left(\frac{1}{1+\frac{1}{|x|}}\cdot sign(x)+1\right).$$

Код 159. Пример использования

```
        double x;
        double y;
        y=MHL_RandomUniform(-100,100);

        //Вызов функции
        x=MHL_NormalizationNumberAll(y);

        //Используем полученный результат
        MHL_ShowNumber (y,"Число", "y");
        //Число:
        //y=-10.4004
        MHL_ShowNumber (x,"Нормированное число", "x");
        //Нормированное число:
        //x=0.0438581
```

6.6.9 MHL_Parity

Функция проверяет четность целого числа.

Код 160. Синтаксис

```
int MHL_Parity(int a);
```

Входные параметры:

a — исходное число.

Возвращаемое значение:

1 — четное;

0 — нечетное.

Код 161. Пример использования

```
int a=MHL_RandomUniformInt(-50,50);

double Result=MHL_Parity(a);

//Используем полученный результат
MHL_ShowNumber(Result,"Четность числа "+MHL_NumberToText(a),"равна");
//Четность числа 2:
//равна=1
```

6.6.10 MHL_SumGeometricSeries

Геометрическая прогрессия. Сумма первых n членов.

Код 162. Синтаксис

```
double MHL_SumGeometricSeries(double u1,double q,int n);
```

Входные параметры:

u1 — начальный член прогрессии;

q — шаг прогрессии;

n — номер последнего члена.

Возвращаемое значение:

Сумма первых n членов.

Код 163. Пример использования

```
double u1=MHL_RandomUniformInt(1,5);
double q=MHL_RandomUniformInt(1,5);
int n=MHL_RandomUniformInt(1,5);

double Sum=MHL_SumGeometricSeries(u1,q,n);

//Используем полученный результат
MHL_ShowNumber(u1,"Первый член последовательности","u1");
//Первый член последовательности:
//u1=4
MHL_ShowNumber(q,"Шаг арифметической прогрессии","q");
//Шаг арифметической прогрессии:
//q=4
MHL_ShowNumber(n,"Номер последнего члена","n");
//Номер последнего члена:
//n=3
MHL_ShowNumber(Sum,"Сумма первых n членов","Sum");
//Сумма первых n членов:
//Sum=84
```

6.6.11 MHL_SumOfArithmeticalProgression

Арифметическая прогрессия. Сумма первых n членов.

Код 164. Синтаксис

```
double MHL_SumOfArithmeticalProgression(double a1,double d,int n);
```

Входные параметры:

a1 — начальный член прогрессии;

d — шаг арифметической прогрессии;

n — номер последнего члена.

Возвращаемое значение:

Сумма первых n членов.

Код 165. Пример использования

```
        double a1=MHL_RandomUniformInt(1,10);
        double d=MHL_RandomUniformInt(1,10);
        int n=MHL_RandomUniformInt(1,10);

        double Sum=MHL_SumOfArithmeticalProgression(a1,d,n);

        //Используем полученный результат
        MHL_ShowNumber(a1,"Первый член последовательности","a1");
        //Первый член последовательности:
        //a1=9
        MHL_ShowNumber(d,"Шаг арифметической прогрессии","d");
        //Шаг арифметической прогрессии:
        //d=6
        MHL_ShowNumber(n,"Номер последнего члена","n");
        //Номер последнего члена:
        //n=9
        MHL_ShowNumber(Sum,"Сумма первых n членов","Sum");
        //Сумма первых n членов:
        //Sum=297
```

6.6.12 MHL_SumOfDigits

Функция подсчитывает сумму цифр любого целого числа.

Код 166. Синтаксис

```
int MHL_SumOfDigits(int a);
```

Входные параметры:

a — целое число.

Возвращаемое значение:

Сумма цифр.

Код 167. Пример использования

```
        int a=MHL_RandomUniformInt(100,30000);

        //Вызов функции
```

```
        int SumOfDigits=MHL_SumOfDigits(a);

        //Используем полученный результат
        MHL_ShowNumber (SumOfDigits,"Сумма цифр числа a="+MHL_NumberToText(a), "равна
            ");
        //Сумма цифр числа a=2069:
        //равна=17
```

6.6.13 TMHL_Abs

Функция возвращает модуль числа.

Код 168. Синтаксис

```
template <class T> T TMHL_Abs(T x);
```

Входные параметры:

x — число.

Возвращаемое значение:

Модуль числа.

Код 169. Пример использования

```
        double x;
        double abs;

        x=MHL_RandomUniform(-10,10);

        //Вызов функции
        abs=TMHL_Abs(x);

        //Используем полученный результат
        MHL_ShowNumber (x,"Число", "x");
        // Число:
        //x=-6.29578
        MHL_ShowNumber (abs,"Модуль", "abs");
        // Модуль:
        //abs=6.29578
```

6.6.14 TMHL_FibonacciNumber

Функция вычисляет число Фибоначчи, заданного номера.

Код 170. Синтаксис

```
template <class T> T TMHL_FibonacciNumber(T n);
```

Входные параметры:

n - номер числа Фибоначчи.

Возвращаемое значение:

Число Фибоначчи, заданного номера.

Код 171. Пример использования

```
int n;
int F;
n=MHL_RandomUniformInt(3,20);

//Вызов функции
F=TMHL_FibonacciNumber(n);

//Используем полученный результат

MHL_ShowNumber (n,"Номер числа", "n");
// Номер числа:
// n=14
MHL_ShowNumber (F,"Число Фибоначчи, заданного номера", "F");
// Число Фибоначчи, заданного номера:
// F=377
```

6.6.15 TMHL_HeavisideFunction

Функция Хевисайда (функция одной переменной).

Код 172. Синтаксис

```
template <class T> T TMHL_HeavisideFunction(T x);
```

Входные параметры:

x — переменная.

Возвращаемое значение:

Значение функции Хевисайда.

Формула:

$$F(x) = \begin{cases} 1, \text{ если } x > 0; \\ 0, \text{ если } x \leq 0. \end{cases}$$

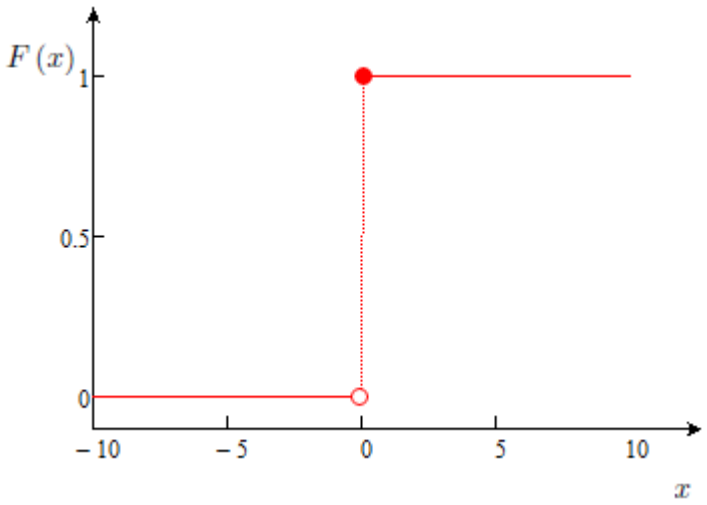

Рисунок 2. График функции

Код 173. Пример использования

```
        double x=MHL_RandomUniform(-50,50);

        double F=TMHL_HeavisideFunction(x);

        //Используем полученный результат
        MHL_ShowNumber(F,"Функция Хэвисайда при a = "+MHL_NumberToText(x),"равна");
        //Функция Хэвисайда при a = -49.7559:
        //равна=0
```

6.6.16 TMHL_Max

Функция возвращает максимальный элемент из двух.

Код 174. Синтаксис

```
template <class T> T TMHL_Max(T a, T b);
```

Входные параметры:

a — первый элемент;

b — первый элемент.

Возвращаемое значение:

Максимальный элемент.

Код 175. Пример использования

```
        int a=MHL_RandomUniformInt(10,100);
        int b=MHL_RandomUniformInt(10,100);

        //Вызов функции
        int Max=TMHL_Max(a,b);

        //Используем полученный результат
        MHL_ShowNumber(Max,"Максимальное среди "+MHL_NumberToText(a)+" и "+
           MHL_NumberToText(b),"равно");
        //Максимальное среди 73 и 44:
        //равно=73
```

6.6.17 TMHL_Min

Функция возвращает минимальный элемент из двух.

Код 176. Синтаксис

```
template <class T> T TMHL_Min(T a, T b);
```

Входные параметры:

a — первый элемент;

b — первый элемент.

Возвращаемое значение:

Минимальный элемент.

Код 177. Пример использования

```
int a=MHL_RandomUniformInt(10,100);
int b=MHL_RandomUniformInt(10,100);

//Вызов функции
int Max=TMHL_Min(a,b);

//Используем полученный результат
MHL_ShowNumber(Max,"Минимальное среди "+MHL_NumberToText(a)+" и "+
    MHL_NumberToText(b),"равно");
//Минимальное среди 79 и 18:
//равно=18
```

6.6.18 TMHL_NumberInterchange

Функция меняет местами значения двух чисел.

Код 178. Синтаксис

```
template <class T> void TMHL_NumberInterchange(T *a, T *b);
```

Входные параметры:

a — первое число;

b — второе число.

Возвращаемое значение:

Отсутствует.

Код 179. Пример использования

```
double a=MHL_RandomUniform(-10,10);
double b=MHL_RandomUniform(-10,10);

MHL_ShowNumber(a,"Было","a");
//Было:
//a=-3.18237
MHL_ShowNumber(b,"Было","b");
//Было:
//b=5.36194

//Вызов функции
TMHL_NumberInterchange(&a,&b);

//Используем полученный результат
MHL_ShowNumber(a,"Стало","a");
//Стало:
//a=5.36194
MHL_ShowNumber(b,"Стало","b");
//Стало:
//b=-3.18237
```

6.6.19 TMHL_PowerOf

Функция возводит произвольное число в целую степень.

Код 180. Синтаксис

```
template <class T> T TMHL_PowerOf(T x, int n);
```

Входные параметры:

x — основание степени;

n — показатель степени.

Возвращаемое значение:

Степень числа.

Код 181. Пример использования

```
        double a=MHL_RandomUniform(-5,5);
        int Degree=MHL_RandomUniformInt(0,20);

        double Result=TMHL_PowerOf(a,Degree);

        //Используем полученный результат
        MHL_ShowNumber(Result,"Число "+MHL_NumberToText(a)+" в степени "+
            MHL_NumberToText(Degree),"равно");
        //Число 3.9624 в степени 4:
        //равно=246.51
```

6.6.20 TMHL_Sign

Функция вычисляет знака числа.

Код 182. Синтаксис

```
template <class T> int TMHL_Sign(T a);
```

Входные параметры:

a — исходное число.

Возвращаемое значение:

0 — если a==0;

1 — если число положительное;

-1 — если число отрицательное.

Код 183. Пример использования

```
        int a=MHL_RandomUniformInt(-5,5);

        //Вызов функции
        int Result=TMHL_Sign(a);

        //Используем полученный результат
```

```
        MHL_ShowNumber(Result,"Знак числа "+MHL_NumberToText(a),"равен");
        //Знак числа -3:
        //равен=-1
```

6.6.21 TMHL_SignNull

Функция вычисляет знака числа. При нуле возвращает 1.

Код 184. Синтаксис

```
template <class T> int TMHL_SignNull(T a);
```

Входные параметры:

a — исходное число.

Возвращаемое значение:

1 — если число неотрицательное;

-1 — если число отрицательное.

Код 185. Пример использования

```
        int a=MHL_RandomUniformInt(-5,5);

        //Вызов функции
        int Result=TMHL_SignNull(a);

        //Используем полученный результат
        MHL_ShowNumber(Result,"Знак числа "+MHL_NumberToText(a),"равен");
        //Знак числа 0:
        //равен=1
```

6.7 Матрицы

6.7.1 TMHL_ColInterchange

Функция переставляет столбцы матрицы.

Код 186. Синтаксис

```
template <class T> void TMHL_ColInterchange(T **VMHL_ResultMatrix, int VMHL_N, int k,
    int l);
```

Входные параметры:

VMHL_ResultMatrix — указатель на исходную матрицу (в ней и сохраняется результат);

VMHL_N — размер массива (число строки);

k,l — номера переставляемых столбцов (нумерация с нуля).

Возвращаемое значение:

Отсутствует.

Код 187. Пример использования

```
int i,j;
int VMHL_N=5;//Размер массива (число строк)
int VMHL_M=5;//Размер массива (число столбцов)
int **Matrix;
Matrix=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) Matrix[i]=new int[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  Matrix[i][j]=MHL_RandomUniformInt(10,100);

MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Случайная матрица", "Matrix");
// Случайная матрица:
//Matrix =
//46   37 90 95 83
//92   58 48 61 16
//31   92 37 64 56
//20   54 84 90 75
//86   79 20 40 69

// номера перставляемых столбцов
int k=MHL_RandomUniformInt(0,5);
int l=MHL_RandomUniformInt(0,5);

//Вызов функции
TMHL_ColInterchange(Matrix,VMHL_N,k,l);

//Используем полученный результат
MHL_ShowNumber (k,"Номер первого столбца","k");
// Номер первого столбца:
//k=4
MHL_ShowNumber (l,"Номер второго столбца","l");
// Номер второго столбца:
//l=0
MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Матрица с персетавленными столбцами", "
    Matrix");
// Матрица с персетавленными столбцами:
//Matrix =
//83   37 90 95 46
//16   58 48 61 92
//56   92 37 64 31
//75   54 84 90 20
//69   79 20 40 86

for (i=0;i<VMHL_N;i++) delete [] Matrix[i];
delete [] Matrix;
```

6.7.2 TMHL_ColToMatrix

Функция копирует в матрицу (двумерный массив) из вектора столбец.

Код 188. Синтаксис

```
template <class T> void TMHL_ColToMatrix(T **VMHL_ResultMatrix, T *b, int VMHL_N, int
    k);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу;

b — указатель на вектор;

VMHL_N — количество строк в матрице и одновременно размер массива b;

k — номер столбца, в который будет происходить копирование (начиная с 0).

Возвращаемое значение:

Отсутствует.

Код 189. Пример использования

```
int i,j;
int VMHL_N=10;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
int *b;
b=new int[VMHL_N];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  a[i][j]=MHL_RandomUniformInt(10,100);
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
//Случайная матрица:
//a =
//13   99 23
//69   44 44
//64   70 72
//14   85 92
//11   40 12
//95   85 81
//82   50 13
//63   82 58
//56   68 89
//51   89 78

for (j=0;j<VMHL_N;j++)
 b[j]=MHL_RandomUniformInt(10,100);

int k=1;//Номер столбца, в который мы копируем

//Вызов функции
TMHL_ColToMatrix(a,b,VMHL_N,k);

//Используем полученный результат
MHL_ShowNumber(k,"Номер столбца, в который мы копируем ","k");
//Номер столбца, в который мы копируем :
//k=1
MHL_ShowVector (b,VMHL_N,"Вектор","b");
//Вектор:
//b =
//35
//92
//90
//41
//17
//24
//11
```

```
        //13
        //23
        //14

        MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Матрица с изменившимся столбцом", "a");
        //Матрица с изменившимся столбцом:
        //a =
        //13   35 23
        //69   92 44
        //64   90 72
        //14   41 92
        //11   17 12
        //95   24 81
        //82   11 13
        //63   13 58
        //56   23 89
        //51   14 78

        for (i=0;i<VMHL_N;i++) delete [] a[i];
        delete [] a;
        delete [] b;
```

6.7.3 TMHL_DeleteColInMatrix

Функция удаляет k столбец из матрицы (начиная с нуля). Все правостоящие столбцы сдвигаются влево на единицу. Последний столбец зануляется.

Код 190. Синтаксис

```
template <class T> void TMHL_DeleteColInMatrix(T **VMHL_ResultMatrix, int k, int
    VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на преобразуемый массив;

k — номер удаляемого столбца;

VMHL_N — размер массива VMHL_ResultMatrix (число строк);

VMHL_M — размер массива VMHL_ResultMatrix (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 191. Пример использования

```
        int i,j;
        int VMHL_N=6;//Размер массива (число строк)
        int VMHL_M=4;//Размер массива (число столбцов)
        double **Matrix;
        Matrix=new double*[VMHL_N];
        for (i=0;i<VMHL_N;i++) Matrix[i]=new double[VMHL_M];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         for (j=0;j<VMHL_M;j++)
          Matrix[i][j]=MHL_RandomUniformInt(10,100);
```

```
        MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Случайная матрица", "Matrix");
        // Случайная матрица:
        //Matrix =
        //39    52 14 31
        //49    49 59 65
        //68    15 12 86
        //91    73 36 32
        //52    31 24 78
        //22    20 33 94

        int k=2;//Удалим второй столбец

        //Вызов функции
        TMHL_DeleteColInMatrix(Matrix,k,VMHL_N,VMHL_M);

        //Используем полученный результат

        MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Матрица с удаленным столбцом", "Matrix"
            );
        // Матрица с удаленным столбцом:
        //Matrix =
        //39    52 31 0
        //49    49 65 0
        //68    15 86 0
        //91    73 32 0
        //52    31 78 0
        //22    20 94 0

        for (i=0;i<VMHL_N;i++) delete [] Matrix[i];
        delete [] Matrix;
```

6.7.4 TMHL_DeleteRowInMatrix

Функция удаляет k строку из матрицы (начиная с нуля). Все нижестоящие строки поднимаются на единицу. Последняя строка зануляется.

Код 192. Синтаксис

```
template <class T> void TMHL_DeleteRowInMatrix(T **VMHL_ResultMatrix, int k, int
    VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на преобразуемый массив;

k — номер удаляемой строки;

VMHL_N — размер массива VMHL_ResultMatrix (число строк);

VMHL_M — размер массива VMHL_ResultMatrix (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 193. Пример использования

```
        int i,j;
        int VMHL_N=6;//Размер массива (число строк)
```

```
int VMHL_M=4;//Размер массива (число столбцов)
double **Matrix;
Matrix=new double*[VMHL_N];
for (i=0;i<VMHL_N;i++) Matrix[i]=new double[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  Matrix[i][j]=MHL_RandomUniformInt(10,100);

MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Случайная матрица", "Matrix");
// Случайная матрица:
//Matrix =
//70   57 44 95
//26   21 60 63
//61   55 27 95
//10   10 43 92
//66   20 51 65
//32   52 63 78

int k=2;//Удалим вторую строку

//Вызов функции
TMHL_DeleteRowInMatrix(Matrix,k,VMHL_N,VMHL_M);

//Используем полученный результат

MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Матрица с удаленной строкой", "Matrix")
    ;
// Матрица с удаленной строкой:
//Matrix =
//70   57 44 95
//26   21 60 63
//10   10 43 92
//66   20 51 65
//32   52 63 78
//0 0  0  0

for (i=0;i<VMHL_N;i++) delete [] Matrix[i];
delete [] Matrix;
```

6.7.5 TMHL_FillMatrix

Функция заполняет матрицу значениями, равных x.

Код 194. Синтаксис

```
template <class T> void TMHL_FillMatrix(T **VMHL_ResultMatrix, int VMHL_N, int VMHL_M
    , T x);
```

Входные параметры:

VMHL_ResultMatrix — указатель на преобразуемый массив;

VMHL_N — размер массива VMHL_ResultMatrix (число строк);

VMHL_M — размер массива VMHL_ResultMatrix (число столбцов);

Возвращаемое значение:

Отсутствует.

Код 195. Пример использования

```
int i;
int VMHL_N=10;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];

int x=MHL_RandomUniformInt(0,10);//заполнитель

//Вызов функции
TMHL_FillMatrix(a,VMHL_N,VMHL_M,x);

//Используем полученный результат
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Заполненная матрица", "a");
//Заполненная матрица:
//a =
//3 3  3
//3 3  3
//3 3  3
//3 3  3
//3 3  3
//3 3  3
//3 3  3
//3 3  3
//3 3  3
//3 3  3

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
```

6.7.6 TMHL_IdentityMatrix

Функция формирует единичную квадратную матрицу.

Код 196. Синтаксис

```
template <class T> void TMHL_IdentityMatrix(T **VMHL_ResultMatrix,int VMHL_N);
```

Входные параметры:

VMHL_ResultMatrix — исходная матрица (в ней и сохраняется результат);

VMHL_N — размер матрицы (число строк и столбцов).

Возвращаемое значение:

Отсутствует.

Код 197. Пример использования

```
int i;
int VMHL_N=5;//Размер массива (число строк и столбцов)
int **Matrix;
Matrix=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) Matrix[i]=new int[VMHL_N];
```

```
//Вызов функции
TMHL_IdentityMatrix(Matrix,VMHL_N);

//Используем полученный результат
MHL_ShowMatrix (Matrix,VMHL_N,VMHL_N,"Единичная матрица", "Matrix");
//Единичная матрица:
//Matrix =
//1 0  0  0  0
//0 1  0  0  0
//0 0  1  0  0
//0 0  0  1  0
//0 0  0  0  1

for (i=0;i<VMHL_N;i++) delete [] Matrix[i];
delete [] Matrix;
```

6.7.7 TMHL_MatrixMinusMatrix

Функция вычитает две матрицы. Или для переопределенной варианта функция вычитает два матрицы и результат записывает в первую матрицу.

Код 198. Синтаксис

```
template <class T> void TMHL_MatrixMinusMatrix(T **a, T **b, T **VMHL_ResultMatrix,
    int VMHL_N, int VMHL_M);
template <class T> void TMHL_MatrixMinusMatrix(T **VMHL_ResultMatrix, T **b, int
    VMHL_N, int VMHL_M);
```

Входные параметры:

a — первая матрица;

b — вторая матрица;

VMHL_ResultMatrix — разница матриц;

VMHL_N — размер матриц (число строк);

VMHL_M — размер матриц (число столбцов).

Возвращаемое значение:

Отсутствует.

Для переопределенного варианта.

Входные параметры:

VMHL_ResultMatrix — первая матрица (в ней и сохраняется разница);

b — вторая матрица;

VMHL_N — размер матриц (число строк);

VMHL_M — размер матриц (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 199. Пример использования

```
int i,j;
int VMHL_N=5;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
int **b;
b=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) b[i]=new int[VMHL_M];
int **c;
c=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) c[i]=new int[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  {
  a[i][j]=MHL_RandomUniformInt(10,20);
  b[i][j]=MHL_RandomUniformInt(10,20);
  }

//Вызов функции
TMHL_MatrixMinusMatrix(a,b,c,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Матрица", "a");
//Матрица:
//a =
//18   19 17
//14   12 11
//10   16 19
//12   18 16
//12   16 11

MHL_ShowMatrix (b,VMHL_N,VMHL_M,"Матрица", "b");
//Матрица:
//b =
//11   19 18
//12   10 13
//11   14 10
//11   17 15
//12   16 10

MHL_ShowMatrix (c,VMHL_N,VMHL_M,"Их разница", "c");
//Их разница:
//c =
//7 0  -1
//2 2  -2
//-1   2  9
//1 1  1
//0 0  1

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
for (i=0;i<VMHL_N;i++) delete [] b[i];
delete [] b;
for (i=0;i<VMHL_N;i++) delete [] c[i];
delete [] c;
```

```
//Для переопределенной функции
VMHL_N=5;//Размер массива (число строк)
VMHL_M=3;//Размер массива (число столбцов)
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
b=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) b[i]=new int[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  {
  a[i][j]=MHL_RandomUniformInt(10,20);
  b[i][j]=MHL_RandomUniformInt(10,20);
  }

MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Матрица", "a");
//Матрица:
//a =
//11   18 11
//19   14 15
//14   13 14
//19   13 12
//19   15 10

//Вызов функции
TMHL_MatrixMinusMatrix(a,b,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (b,VMHL_N,VMHL_M,"Матрица", "b");
//Матрица:
//b =
//12   13 18
//14   12 14
//12   14 19
//18   16 16
//16   17 19

MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Теперь матрица a", "a");
//Теперь матрица a:
//a =
//-1   5  -7
//5 2  1
//2 -1 -5
//1 -3 -4
//3 -2 -9

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
for (i=0;i<VMHL_N;i++) delete [] b[i];
delete [] b;
```

6.7.8 TMHL_MatrixMultiplyMatrix

Функция перемножает матрицы.

Код 200. Синтаксис

```
template <class T> void TMHL_MatrixMultiplyMatrix(T **a, T **b, T **VMHL_ResultMatrix
    , int VMHL_N, int VMHL_M, int VMHL_S);
```

Входные параметры:

a — первый сомножитель, VMHL_N x VMHL_M;

b — второй сомножитель, VMHL_M x VMHL_S;

VMHL_ResultMatrix — произведение матриц (сюда записывается результат), VMHL_N x VMHL_S;

VMHL_N — число строк в матрице a;

VMHL_M — число столбцов в матрице a и строк в матрице b;

VMHL_S — число столбцов в матрице b.

Возвращаемое значение:

Отсутствует.

Код 201. Пример использования

```
        int i;
        int VMHL_N=3;
        int VMHL_M=5;
        int VMHL_S=4;
        int **a;
        a=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
        int **b;
        b=new int*[VMHL_M];
        for (i=0;i<VMHL_M;i++) b[i]=new int[VMHL_S];
        int **c;
        c=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) c[i]=new int[VMHL_S];
        TMHL_RandomIntMatrix(a,0,10,VMHL_N,VMHL_M);
        TMHL_RandomIntMatrix(b,0,10,VMHL_M,VMHL_S);

        //Вызов функции
        TMHL_MatrixMultiplyMatrix (a, b, c, VMHL_N, VMHL_M, VMHL_S);

        //Используем полученный результат
        MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
        //Случайная матрица:
        //a =
        //3 0  4  4  5
        //9 4  3  4  4
        //8 0  1  9  8

        MHL_ShowMatrix (b,VMHL_M,VMHL_S,"Случайная матрица", "b");
        // Случайная матрица:
        //b =
        //6 6  6  3
        //4 2  1  2
        //6 9  6  3
        //1 1  8  2
        //6 8  0  9

        MHL_ShowMatrix (c,VMHL_N,VMHL_S,"Произведение", "c");
```

```
        // Произведение:
        //c =
        //76   98 74 74
        //116  125   108   88
        //111  130   126   117

        for (i=0;i<VMHL_N;i++) delete [] a[i];
        delete [] a;
        for (i=0;i<VMHL_M;i++) delete [] b[i];
        delete [] b;
        for (i=0;i<VMHL_N;i++) delete [] c[i];
        delete [] c;
```

6.7.9 TMHL_MatrixMultiplyMatrixT

Функция умножает матрицу на транспонированную матрицу.

Код 202. Синтаксис

```
template <class T> void TMHL_MatrixMultiplyMatrixT(T **a, T **b, T **
    VMHL_ResultMatrix, int VMHL_N, int VMHL_M, int VMHL_S);
```

Входные параметры:

a — первый сомножитель, VMHL_N x VMHL_M;

b — второй сомножитель (матрица, которую мы транспонируем), VMHL_S x VMHL_M;

VMHL_ResultMatrix — произведение матриц (сюда записывается результат), VMHL_N x VMHL_S;

VMHL_N — число строк в матрице a;

VMHL_M — число столбцов в матрице a и столбцов в матрице b;

VMHL_S — число строк в матрице b.

Возвращаемое значение:

Отсутствует.

Код 203. Пример использования

```
        int i;
        int VMHL_N=3;
        int VMHL_M=5;
        int VMHL_S=4;
        int **a;
        a=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
        int **b;
        b=new int*[VMHL_S];
        for (i=0;i<VMHL_S;i++) b[i]=new int[VMHL_M];
        int **c;
        c=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) c[i]=new int[VMHL_S];
        TMHL_RandomIntMatrix(a,0,10,VMHL_N,VMHL_M);
        TMHL_RandomIntMatrix(b,0,10,VMHL_S,VMHL_M);
```

```
        //Вызов функции
        TMHL_MatrixMultiplyMatrixT (a, b, c, VMHL_N, VMHL_M, VMHL_S);

        //Используем полученный результат
        MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
        //Случайная матрица:
        //a =
        //9 8  5  2  1
        //1 9  3  4  8
        //9 9  3  0  3

        MHL_ShowMatrix (b,VMHL_S,VMHL_M,"Случайная матрица", "b");
        // Случайная матрица:
        //b =
        //3 7  3  0  8
        //9 8  0  6  9
        //0 2  5  6  5
        //8 7  9  2  3

        MHL_ShowMatrix (c,VMHL_N,VMHL_S,"Произведение", "c");
        // Произведение:
        //c =
        //106  166   58 180
        //139  177   97 130
        //123  180   48 171

        for (i=0;i<VMHL_N;i++) delete [] a[i];
        delete [] a;
        for (i=0;i<VMHL_S;i++) delete [] b[i];
        delete [] b;
        for (i=0;i<VMHL_N;i++) delete [] c[i];
        delete [] c;
```

6.7.10 TMHL_MatrixMultiplyNumber

Функция умножает матрицу на число.

Код 204. Синтаксис

```
template <class T> void TMHL_MatrixMultiplyNumber(T **VMHL_ResultMatrix, int VMHL_N,
    int VMHL_M, T Number);
```

Входные параметры:

VMHL_ResultMatrix — указатель на исходную матрицу (в ней и сохраняется результат);

VMHL_N — размер матрицы (число строк);

VMHL_M — размер матрицы (число столбцов);

Number — число, на которое умножается матрица.

Возвращаемое значение:

Отсутствует.

Код 205. Пример использования

```
        int i,j;
```

```
int VMHL_N=5;//Размер массива (число строк)
int VMHL_M=5;//Размер массива (число столбцов)
int **Matrix;
Matrix=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) Matrix[i]=new int[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  Matrix[i][j]=MHL_RandomUniformInt(10,100);

MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Случайная матрица", "Matrix");
//Случайная матрица:
//Matrix =
//77   34 14 83 30
//31   15 87 68 20
//52   11 49 92 95
//77   29 96 50 90
//10   47 40 49 20

int Number=MHL_RandomUniformInt(-10,10);

//Вызов функции
TMHL_MatrixMultiplyNumber(Matrix,VMHL_N,VMHL_M,Number);

//Используем полученный результат
MHL_ShowNumber (Number,"Число, на которое умножается матрица","Number");
//Число, на которое умножается матрица:
//Number=4
MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Матрица умноженное на число", "Matrix")
    ;
//Матрица умноженное на число:
//Matrix =
//308  136   56 332   120
//124  60 348   272   80
//208  44 196   368   380
//308  116   384   200   360
//40   188   160   196   80

for (i=0;i<VMHL_N;i++) delete [] Matrix[i];
delete [] Matrix;
```

6.7.11 TMHL_MatrixPlusMatrix

Функция суммирует две матрицы. Или для переопределенной варианта функция суммирует два матрицы и результат записывает в первую матрицу.

Код 206. Синтаксис

```
template <class T> void TMHL_MatrixPlusMatrix(T **a, T **b, T **VMHL_ResultMatrix,
   int VMHL_N, int VMHL_M);
template <class T> void TMHL_MatrixPlusMatrix(T **VMHL_ResultMatrix, T **b, int
   VMHL_N, int VMHL_M);
```

Входные параметры:

a — первая матрица;

b — вторая матрица;

VMHL_ResultMatrix — сумма матриц;

VMHL_N — размер матриц (число строк);

VMHL_M — размер матриц (число столбцов).

Возвращаемое значение:

Отсутствует.

Для переопределенного варианта.

Входные параметры:

VMHL_ResultMatrix — первая матрица (в ней и сохраняется сумма);

b — вторая матрица;

VMHL_N — размер матриц (число строк);

VMHL_M — размер матриц (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 207. Пример использования

```
int i,j;
int VMHL_N=5;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
int **b;
b=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) b[i]=new int[VMHL_M];
int **c;
c=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) c[i]=new int[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  {
  a[i][j]=MHL_RandomUniformInt(10,20);
  b[i][j]=MHL_RandomUniformInt(10,20);
  }

//Вызов функции
TMHL_MatrixPlusMatrix(a,b,c,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Матрица", "a");
//Матрица:
//a =
//18   15 15
//15   11 17
//19   14 10
//17   18 18
//19   15 16

MHL_ShowMatrix (b,VMHL_N,VMHL_M,"Матрица", "b");
```

```
//Матрица:
//b =
//17   15 15
//16   18 10
//17   12 15
//12   16 13
//15   14 10

MHL_ShowMatrix (c,VMHL_N,VMHL_M,"Их сумма", "c");
//Их сумма:
//c =
//35   30 30
//31   29 27
//36   26 25
//29   34 31
//34   29 26

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
for (i=0;i<VMHL_N;i++) delete [] b[i];
delete [] b;
for (i=0;i<VMHL_N;i++) delete [] c[i];
delete [] c;

//Для переопределенной функции
VMHL_N=5;//Размер массива (число строк)
VMHL_M=3;//Размер массива (число столбцов)
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
b=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) b[i]=new int[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  {
  a[i][j]=MHL_RandomUniformInt(10,20);
  b[i][j]=MHL_RandomUniformInt(10,20);
  }

MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Матрица", "a");
//Матрица:
//a =
//18   12 12
//19   17 12
//19   17 17
//11   10 17
//11   19 10

//Вызов функции
TMHL_MatrixPlusMatrix(a,b,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (b,VMHL_N,VMHL_M,"Матрица", "b");
//Матрица:
//b =
//10   10 16
//10   18 18
//15   13 17
//13   11 14
//16   13 11
```

```
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Теперь матрица a", "a");
//Теперь матрица a:
//a =
//28   22 28
//29   35 30
//34   30 34
//24   21 31
//27   32 21

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
for (i=0;i<VMHL_N;i++) delete [] b[i];
delete [] b;
```

6.7.12 TMHL_MatrixT

Функция транспонирует матрицу.

Код 208. Синтаксис

```
template <class T> void TMHL_MatrixT(T **a, T **VMHL_ResultMatrix, int VMHL_N, int
    VMHL_M);
```

Входные параметры:

a — исходная матрица, (VMHL_N x VMHL_M);

VMHL_ResultMatrix — транспонированная матрица, (VMHL_M x VMHL_N);

VMHL_N — размер матрицы (число строк) в матрице a;

VMHL_M — размер матрицы (число столбцов) в матрице a.

Возвращаемое значение:

Отсутствует.

Код 209. Пример использования

```
int i,j;
int VMHL_N=5;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **Matrix;
Matrix=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) Matrix[i]=new int[VMHL_M];
int **MatrixT;
MatrixT=new int*[VMHL_M];
for (i=0;i<VMHL_M;i++) MatrixT[i]=new int[VMHL_N];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  Matrix[i][j]=MHL_RandomUniformInt(10,100);

//Вызов функции
TMHL_MatrixT(Matrix,MatrixT,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Матрица", "Matrix");
```

```
        //Матрица:
        //Matrix =
        //26    64 62
        //70    49 43
        //50    41 50
        //76    75 81
        //26    72 24

        MHL_ShowMatrix (MatrixT,VMHL_M,VMHL_N,"Транспонированная матрица", "MatrixT")
            ;
        // Транспонированная матрица:
        //MatrixT =
        //26    70 50 76 26
        //64    49 41 75 72
        //62    43 50 81 24

        for (i=0;i<VMHL_N;i++) delete [] Matrix[i];
        delete [] Matrix;
        for (i=0;i<VMHL_M;i++) delete [] MatrixT[i];
        delete [] MatrixT;
```

6.7.13 TMHL_MatrixTMultiplyMatrix

Функция умножает транспонированную матрицу на матрицу.

Код 210. Синтаксис

```
template <class T> void TMHL_MatrixTMultiplyMatrix(T **a, T **b, T **
    VMHL_ResultMatrix, int VMHL_N, int VMHL_M, int VMHL_S);
```

Входные параметры:

a — первый сомножитель (матрица, которую мы транспонируем), VMHL_M x VMHL_N;

b — второй сомножитель, VMHL_M x VMHL_S;

VMHL_ResultMatrix — произведение матриц (сюда записывается результат), VMHL_N x VMHL_S;

VMHL_N — число столбцов в матрице a;

VMHL_M — число строк в матрице a и строк в матрице b;

VMHL_S — число столбцов в матрице b.

Возвращаемое значение:

Отсутствует.

Код 211. Пример использования

```
        int i;
        int VMHL_N=3;
        int VMHL_M=5;
        int VMHL_S=4;
        int **a;
        a=new int*[VMHL_M];
        for (i=0;i<VMHL_M;i++) a[i]=new int[VMHL_N];
        int **b;
```

```
b=new int*[VMHL_M];
for (i=0;i<VMHL_M;i++) b[i]=new int[VMHL_S];
int **c;
c=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) c[i]=new int[VMHL_S];
TMHL_RandomIntMatrix(a,0,10,VMHL_M,VMHL_N);
TMHL_RandomIntMatrix(b,0,10,VMHL_M,VMHL_S);

//Вызов функции
TMHL_MatrixTMultiplyMatrix (a, b, c, VMHL_N, VMHL_M, VMHL_S);

//Используем полученный результат
MHL_ShowMatrix (a,VMHL_M,VMHL_N,"Случайная матрица", "a");
// Случайная матрица:
//a =
//6 0  1
//6 5  9
//7 2  0
//3 1  5
//3 8  8

MHL_ShowMatrix (b,VMHL_M,VMHL_S,"Случайная матрица", "b");
// Случайная матрица:
//b =
//6 7  1  0
//7 6  0  0
//5 6  0  0
//9 7  9  3
//5 7  0  1

MHL_ShowMatrix (c,VMHL_N,VMHL_S,"Произведение", "c");
// Произведение:
//c =
//155  162   33 12
//94   105   9  11
//154  152   46 23

for (i=0;i<VMHL_M;i++) delete [] a[i];
delete [] a;
for (i=0;i<VMHL_M;i++) delete [] b[i];
delete [] b;
for (i=0;i<VMHL_N;i++) delete [] c[i];
delete [] c;
```

6.7.14 TMHL_MatrixToCol

Функция копирует из матрицы (двумерного массива) в вектор столбец.

Код 212. Синтаксис

```
template <class T> void TMHL_MatrixToCol(T **a, T *VMHL_ResultVector, int VMHL_N, int
    k);
```

Входные параметры:

a — указатель на матрицу;

VMHL_ResultVector — указатель на вектор;

VMHL_N — количество строк в матрице и одновременно размер массива b;

k — номер копируемого столбца (начиная с 0).

Возвращаемое значение:

Отсутствует.

Код 213. Пример использования

```
int i,j;
int VMHL_N=10;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
int *b;
b=new int[VMHL_N];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  a[i][j]=MHL_RandomUniformInt(10,100);

int k=1;//Номер копируемого столбца

//Вызов функции
TMHL_MatrixToCol(a,b,VMHL_N,k);

//Используем полученный результат
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
//Случайная матрица:
//a =
//98   39 35
//18   30 95
//68   81 59
//43   20 23
//94   40 14
//17   36 84
//98   13 69
//11   94 63
//62   80 22
//27   17 58

MHL_ShowNumber(k,"Номер копируемого столбца ","k");
//Номер копируемого столбца :
//k=1
MHL_ShowVector (b,VMHL_N,"Вектор, в который скопировали столбец","b");
//Вектор, в который скопировали столбец:
//b =
//39
//30
//81
//20
//40
//36
//13
//94
//80
//17

for (i=0;i<VMHL_N;i++) delete [] a[i];
```

```
        delete [] a;
        delete [] b;
```

6.7.15 TMHL_MatrixToMatrix

Функция копирует содержимое матрицы (двумерного массива) a в массив VMHL_ResultMatrix.

Код 214. Синтаксис

```
template <class T> void TMHL_MatrixToMatrix(T **a, T **VMHL_ResultMatrix, int VMHL_N,
    int VMHL_M);
```

Входные параметры:

a — указатель на исходный массив;

VMHL_ResultMatrix — указатель на массив в который производится запись;

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 215. Пример использования

```
        int i,j;
        int VMHL_N=10;//Размер массива (число строк)
        int VMHL_M=3;//Размер массива (число столбцов)
        double **a;
        a=new double*[VMHL_N];
        for (i=0;i<VMHL_N;i++) a[i]=new double[VMHL_M];
        double **b;
        b=new double*[VMHL_N];
        for (i=0;i<VMHL_N;i++) b[i]=new double[VMHL_M];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         for (j=0;j<VMHL_M;j++)
          a[i][j]=MHL_RandomUniformInt(10,100);

        //Вызов функции
        TMHL_MatrixToMatrix(a,b,VMHL_N,VMHL_M);

        //Используем полученный результат
        MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
        //Случайная матрица:
        //a =
        //82   55 19
        //38   82 91
        //68   67 50
        //82   62 63
        //24   41 69
        //16   47 29
        //18   92 63
        //11   29 30
        //71   49 64
```

```
        //11   95 38

        MHL_ShowMatrix (b,VMHL_N,VMHL_M,"Теперь b равна a", "b");
        //Теперь b равна a:
        //b =
        //82   55 19
        //38   82 91
        //68   67 50
        //82   62 63
        //24   41 69
        //16   47 29
        //18   92 63
        //11   29 30
        //71   49 64
        //11   95 38

        for (i=0;i<VMHL_N;i++) delete [] a[i];
        delete [] a;
        for (i=0;i<VMHL_N;i++) delete [] b[i];
        delete [] b;
```

6.7.16 TMHL_MatrixToRow

Функция копирует из матрицы (двумерного массива) в вектор строку.

Код 216. Синтаксис

```
template <class T> void TMHL_MatrixToRow(T **a, T *VMHL_ResultVector, int k, int
    VMHL_M);
```

Входные параметры:

a — указатель на матрицу;

VMHL_ResultVector — указатель на вектор;

k — номер копируемой строки (начиная с 0);

VMHL_M — количество столбцов в матрице и одновременно размер массива VMHL_ResultVector.

Возвращаемое значение:

Отсутствует.

Код 217. Пример использования

```
        int i,j;
        int VMHL_N=10;//Размер массива (число строк)
        int VMHL_M=3;//Размер массива (число столбцов)
        int **a;
        a=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
        int *b;
        b=new int[VMHL_M];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         for (j=0;j<VMHL_M;j++)
          a[i][j]=MHL_RandomUniformInt(10,100);
```

```
        int k=1;//Номер копируемой строки

        //Вызов функции
        TMHL_MatrixToRow(a,b,k,VMHL_M);

        //Используем полученный результат
        MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
        //Случайная матрица:
        //a =
        //31    57 29
        //69    75 13
        //85    14 75
        //78    91 11
        //83    23 94
        //79    48 31
        //43    18 70
        //80    18 15
        //38    95 78
        //16    90 69

        MHL_ShowNumber(k,"Номер копируемой строки ","k");
        //Номер копируемой строки :
        //k=1
        MHL_ShowVector (b,VMHL_M,"Вектор, в который скопировали строку","b");
        //Вектор, в который скопировали строку:
        //b =
        //69
        //75
        //13

        for (i=0;i<VMHL_N;i++) delete [] a[i];
        delete [] a;
        delete [] b;
```

6.7.17 TMHL_MaximumOfMatrix

Функция ищет максимальный элемент в матрице (двумерном массиве).

Код 218. Синтаксис

```
template <class T> T TMHL_MaximumOfMatrix(T **a, int VMHL_N, int VMHL_M);
```

Входные параметры:

a — указатель на матрицу;

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение:

Максимальный элемент.

Код 219. Пример использования

```
        int i,j;
        int VMHL_N=10;//Размер массива (число строк)
```

```
int VMHL_M=3;//Размер массива (число столбцов)
double **Matrix;
Matrix=new double*[VMHL_N];
for (i=0;i<VMHL_N;i++) Matrix[i]=new double[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  Matrix[i][j]=MHL_RandomUniformInt(10,100);

//Вызов функции
double Maximum=TMHL_MaximumOfMatrix(Matrix,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Случайная матрица", "Matrix");
//Случайная матрица:
//Matrix =
//25   42 79
//99   34 34
//16   80 41
//12   95 78
//67   27 14
//29   20 93
//57   66 17
//52   38 42
//31   96 27
//39   77 50

MHL_ShowNumber(Maximum,"Максимальный элемент","Maximum");
//Максимальный элемент:
//Maximum=96

for (i=0;i<VMHL_N;i++) delete [] Matrix[i];
delete [] Matrix;
```

6.7.18 TMHL_MinimumOfMatrix

Функция ищет минимальный элемент в матрице (двумерном массиве).

Код 220. Синтаксис

```
template <class T> T TMHL_MinimumOfMatrix(T **a, int VMHL_N, int VMHL_M);
```

Входные параметры:

a — указатель на матрицу;

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение:

Минимальный элемент.

Код 221. Пример использования

```
int i,j;
int VMHL_N=10;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
```

```
double **Matrix;
Matrix=new double*[VMHL_N];
for (i=0;i<VMHL_N;i++) Matrix[i]=new double[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  Matrix[i][j]=MHL_RandomUniformInt(10,100);

//Вызов функции
double Minimum=TMHL_MinimumOfMatrix(Matrix,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Случайная матрица", "Matrix");
//Случайная матрица:
//Matrix =
//29   64 95
//55   25 36
//73   31 62
//54   19 22
//29   78 48
//24   40 46
//82   13 90
//66   23 14
//44   45 56
//73   92 16

MHL_ShowNumber(Minimum,"Минимальный элемент","Minimum");
//Минимальный элемент:
//Minimum=13

for (i=0;i<VMHL_N;i++) delete [] Matrix[i];
delete [] Matrix;
```

6.7.19 TMHL_MixingRowsInOrder

Функция меняет строки матрицы в порядке, указанным в массиве b.

Код 222. Синтаксис

```
template <class T> void TMHL_MixingRowsInOrder(T **VMHL_ResultMatrix, int *b, int
   VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу, в которой меняем порядок строк;

b — вектор, в котором записаны номера, под которыми должны стать строки в матрице (от 0 до VMHL_N—1);

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 223. Пример использования

```
int i;
int VMHL_N=7;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)

int *b;
b=new int[VMHL_N];

int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
//Заполним случайными числами
TMHL_RandomIntMatrix(a,10,100,VMHL_N,VMHL_M);
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
//Случайная матрица:
//a =
//49    65 82
//92    73 27
//10    72 80
//87    62 12
//82    11 75
//15    75 94
//56    96 39

//Первончальный порядок
TMHL_OrdinalVectorZero(b,VMHL_N);
//Перемешаем
TMHL_MixingVector(b,0.5,VMHL_N);

//Вызов функции
TMHL_MixingRowsInOrder(a,b,VMHL_N,VMHL_M);

//Используем полученный результат

MHL_ShowVector (b,VMHL_N,"Номера нового порядка", "b");
//Номера нового порядка:
//b =
//5
//0
//1
//4
//6
//2
//3

MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица с новым порядком строк", "
    a");
//Случайная матрица с новым порядком строк:
//a =
//92    73 27
//10    72 80
//15    75 94
//56    96 39
//87    62 12
//49    65 82
//82    11 75

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
delete [] b;
```

6.7.20 TMHL_NumberOfDifferentValuesInMatrix

Функция подсчитывает число различных значений в матрице.

Код 224. Синтаксис

```
template <class T> int TMHL_NumberOfDifferentValuesInMatrix(T **a, int VMHL_N, int
    VMHL_M);
```

Входные параметры:

a — указатель на матрица;

VMHL_N — размер массива a (строки);

VMHL_M — размер массива a (столбцы).

Возвращаемое значение:

Отсутствует.

Примечание:

Алгоритм очень топорный и медленный.

Код 225. Пример использования

```
        int i,j;
        int VMHL_N=5;//Размер массива (число строк)
        int VMHL_M=3;//Размер массива (число столбцов)
        int **a;
        a=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         for (j=0;j<VMHL_M;j++)
          a[i][j]=MHL_RandomUniformInt(0,50);

        //Вызов функции
        int NumberOfDifferent=TMHL_NumberOfDifferentValuesInMatrix(a,VMHL_N,VMHL_M);

        //Используем полученный результат
        MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
        //Случайная матрица:
        //a =
        //7 3  27
        //31   30 10
        //37   34 49
        //45   26 12
        //26   28 0

        MHL_ShowNumber (NumberOfDifferent,"Число различных значений в матрице", "
            NumberOfDifferent");
        //Число различных значений в матрице:
        //NumberOfDifferent=14
        for (i=0;i<VMHL_N;i++) delete [] a[i];
        delete [] a;
```

6.7.21 TMHL_RowInterchange

Функция переставляет строки матрицы.

Код 226. Синтаксис

```
template <class T> void TMHL_MatrixToRow(T **a, T *VMHL_ResultVector, int k, int
    VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на исходную матрицу (в ней и сохраняется результат);

VMHL_M — размер массива (число столбцов);

k,l — номера переставляемых строк (нумерация с нуля).

Возвращаемое значение:

Отсутствует.

Код 227. Пример использования

```
        int i,j;
        int VMHL_N=5;//Размер массива (число строк)
        int VMHL_M=5;//Размер массива (число столбцов)
        int **Matrix;
        Matrix=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) Matrix[i]=new int[VMHL_M];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         for (j=0;j<VMHL_M;j++)
          Matrix[i][j]=MHL_RandomUniformInt(10,100);

        MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Случайная матрица", "Matrix");
        //Случайная матрица:
        //Matrix =
        //64    41 93 98 45
        //19    55 31 38 44
        //38    78 39 44 86
        //28    54 39 14 72
        //31    99 64 49 63

        // номера перставляемых строк
        int k=MHL_RandomUniformInt(0,5);
        int l=MHL_RandomUniformInt(0,5);

        //Вызов функции
        TMHL_RowInterchange(Matrix,VMHL_M,k,l);

        //Используем полученный результат
        MHL_ShowNumber (k,"Номер первой строки","k");
        //Номер первой строки:
        //k=4
        MHL_ShowNumber (l,"Номер второй строки","l");
        //Номер второй строки:
        //l=3
        MHL_ShowMatrix (Matrix,VMHL_N,VMHL_M,"Матрица с персетавленными строками", "
            Matrix");
        //Матрица с персетавленными строками:
        //Matrix =
```

```
//64   41 93 98 45
//19   55 31 38 44
//38   78 39 44 86
//31   99 64 49 63
//28   54 39 14 72

for (i=0;i<VMHL_N;i++) delete [] Matrix[i];
delete [] Matrix;
```

6.7.22 TMHL_RowToMatrix

Функция копирует в матрицу (двумерный массив) из вектора строку.

Код 228. Синтаксис

```
template <class T> void TMHL_RowToMatrix(T **VMHL_ResultMatrix, T *b, int k, int
    VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу;

b — указатель на вектор;

k — номер строки, в которую будет происходить копирование (начиная с 0);

VMHL_M — количество столбцов в матрице и одновременно размер массива b.

Возвращаемое значение:

Отсутствует.

Код 229. Пример использования

```
int i,j;
int VMHL_N=10;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
int *b;
b=new int[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  a[i][j]=MHL_RandomUniformInt(10,100);
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
//Случайная матрица:
//a =
//53   15 56
//47   85 84
//82   56 58
//24   34 53
//42   34 20
//76   46 24
//93   17 17
//73   31 26
//29   63 20
//84   83 98
```

```
for (j=0;j<VMHL_M;j++)
 b[j]=MHL_RandomUniformInt(10,100);

int k=1;//Номер строки, в которую мы копируем

//Вызов функции
TMHL_RowToMatrix(a,b,k,VMHL_M);

//Используем полученный результат
MHL_ShowNumber(k,"Номер строки, в которую мы копируем ","k");
//Номер строки, в которую мы копируем :
//k=1
MHL_ShowVector (b,VMHL_M,"Вектор","b");
//Вектор:
//b =
//92
//89
//11

MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Матрица с изменившейся строкой", "a");
//Матрица с изменившейся строкой:
//a =
//53   15 56
//92   89 11
//82   56 58
//24   34 53
//42   34 20
//76   46 24
//93   17 17
//73   31 26
//29   63 20
//84   83 98

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
delete [] b;
```

6.7.23 TMHL_SumMatrix

Функция вычисляет сумму элементов матрицы.

Код 230. Синтаксис

```
template <class T> T TMHL_SumMatrix(T **a,int VMHL_N,int VMHL_M);
```

Входные параметры:

a — указатель на исходный массив;

VMHL_N — размер массива a (число строк);

VMHL_M — размер массива a (число столбцов).

Возвращаемое значение:

Сумма элементов матрицы.

Код 231. Пример использования

```
int i,j;
int VMHL_N=10;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  a[i][j]=MHL_RandomUniformInt(10,100);

//Вызов функции
int SumMatrix=TMHL_SumMatrix(a,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
//Случайная матрица:
//a =
//93   11 72
//58   74 66
//39   16 46
//87   23 76
//85   60 13
//34   43 63
//11   99 20
//77   93 70
//68   32 65
//36   74 35

MHL_ShowNumber (SumMatrix,"Её сумма", "SumVector");
//Её сумма:
//SumVector=1639

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
```

6.7.24 TMHL_ZeroMatrix

Функция зануляет матрицу.

Код 232. Синтаксис

```
template <class T> void TMHL_ZeroMatrix(T **VMHL_ResultMatrix,int VMHL_N,int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на преобразуемый массив;

VMHL_N — размер массива VMHL_ResultMatrix (число строк);

VMHL_M — размер массива VMHL_ResultMatrix (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 233. Пример использования

```
int i,j;
int VMHL_N=10;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  a[i][j]=MHL_RandomUniformInt(10,100);

//Вызов функции
int SumMatrix=TMHL_SumMatrix(a,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
//Случайная матрица:
//a =
//93   11 72
//58   74 66
//39   16 46
//87   23 76
//85   60 13
//34   43 63
//11   99 20
//77   93 70
//68   32 65
//36   74 35

MHL_ShowNumber (SumMatrix,"Её сумма", "SumVector");
//Её сумма:
//SumVector=1639

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
```

6.8 Метрика

6.8.1 TMHL_Chebychev

Функция вычисляет расстояние Чебышева.

Код 234. Синтаксис

```
template <class T> T TMHL_Chebychev(T *x, T *y, int VMHL_N);
```

Входные параметры:

x — указатель на первый вектор;

y — указатель на второй вектор;

VMHL_N — размер массивов.

Возвращаемое значение:

Значение метрики расстояние Чебышева.

Формула:

$$S\left(\bar{x},\bar{y}\right)=\max_{i\in\overline{1,n}}\left(\left|x_i-y_i\right|\right).$$

Код 235. Пример использования

```
int VMHL_N=5;//Размер массива
double *x;
x=new double[VMHL_N];
double *y;
y=new double[VMHL_N];
//Заполним случайными числами
MHL_RandomRealVector (x,0,10,VMHL_N);
MHL_RandomRealVector (y,0,10,VMHL_N);

//Вызов функции
double metric=TMHL_Chebychev(x,y,VMHL_N);

//Используем полученный результат
MHL_ShowVector (x,VMHL_N,"Первый массив", "x");
//Первый массив:
//x =
//7.9245
//7.28699
//6.24054
//1.12152
//7.65442

MHL_ShowVector (y,VMHL_N,"Второй массив", "y");
//Второй массив:
//y =
//0.324097
//3.12164
//4.47266
//9.70062
//5.64117

MHL_ShowNumber (metric,"Значение метрики расстояние Чебышева", "metric");
//Значение метрики расстояние Чебышева:
//metric=8.5791

delete [] x;
delete [] y;
```

6.8.2 TMHL_CityBlock

Функция вычисляет манхэттенское расстояние между двумя массивами.

Код 236. Синтаксис

```
template <class T> T TMHL_CityBlock(T *x, T *y, int VMHL_N);
```

Входные параметры:

x — указатель на первый вектор;

y — указатель на второй вектор;

VMHL_N — размер массивов.

Возвращаемое значение:

Значение метрики манхэттенское расстояние.

Формула:

$$S\left(\bar{x},\bar{y}\right)=\sum_{i=1}^{n}\left|x_i-y_i\right|.$$

Код 237. Пример использования

```
int i;
int VMHL_N=5;//Размер массива
int *x;
x=new int[VMHL_N];
int *y;
y=new int[VMHL_N];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 {
 x[i]=MHL_RandomUniformInt(0,5);
 y[i]=MHL_RandomUniformInt(0,5);
 }

//Вызов функции
int metric=TMHL_CityBlock(x,y,VMHL_N);

//Используем полученный результат
MHL_ShowVector (x,VMHL_N,"Первый массив", "x");
//Первый массив:
//x =
//4
//1
//3
//3
//0

MHL_ShowVector (y,VMHL_N,"Второй массив", "y");
//Второй массив:
//y =
//3
//4
//1
//4
//1

MHL_ShowNumber (metric,"Значение метрики манхэттенское расстояние", "metric")
    ;
// Значение метрики манхэттенское расстояние:
//metric=8

delete [] x;
delete [] y;
```

6.8.3 TMHL_Euclid

Функция вычисляет евклидово расстояние.

Код 238. Синтаксис

```
template <class T> T TMHL_Euclid(T *x, T *y, int VMHL_N);
```

Входные параметры:

x — указатель на первый вектор;

y — указатель на второй вектор;

VMHL_N — размер массивов.

Возвращаемое значение:

Значение метрики евклидово расстояние.

Формула:

$$S\left(\bar{x},\bar{y}\right)=\sqrt{\sum_{i=1}^{n}\left(x_i-y_i\right)^2}.$$

Код 239. Пример использования

```
int VMHL_N=5;//Размер массива
double *x;
x=new double[VMHL_N];
double *y;
y=new double[VMHL_N];
//Заполним случайными числами
MHL_RandomRealVector (x,0,10,VMHL_N);
MHL_RandomRealVector (y,0,10,VMHL_N);

//Вызов функции
double metric=TMHL_Euclid(x,y,VMHL_N);

//Используем полученный результат
MHL_ShowVector (x,VMHL_N,"Первый массив", "x");
//Первый массив:
//x =
//3.15491
//4.04266
//2.63519
//9.94141
//3.2193

MHL_ShowVector (y,VMHL_N,"Второй массив", "y");
//Второй массив:
//y =
//9.4516
//2.59064
//2.56348
//4.78271
//5.78705

MHL_ShowNumber (metric,"Значение метрики евклидово расстояние", "metric");
```

```
        //Значение метрики евклидово расстояние:
        //metric=8.65837

        delete [] x;
        delete [] y;
```

6.9 Оптимизация

6.9.1 MHL_BinaryMonteCarloAlgorithm

Метод Монте-Карло (Monte-Carlo). Простейший метод оптимизации на бинарных строках. В простонародье его называют "методом научного тыка". Алгоритм оптимизации. Ищет максимум функции пригодности FitnessFunction.

Код 240. Синтаксис

```
int MHL_BinaryMonteCarloAlgorithm(int *Parameters, double (*FitnessFunction)(int*,int
    ), int *VMHL_ResultVector, double *VMHL_Result);
```

Входные параметры:

Parameters:

0 — длина бинарной строки (определается задачей оптимизации, что мы решаем);

1 — число вычислений функции пригодности (CountOfFitness);

FitnessFunction - указатель на функцию пригодности (не целевая функция, а именно функция пригодности);

VMHL_ResultVector - найденное решение (бинарный вектор);

VMHL_Result - значение функции в точке, определенной вектором VMHL_ResultVector.

Возвращаемое значение:

1 — завершил работу без ошибок. Всё хорошо.

0 — возникли при работе ошибки. Скорее всего в этом случае в VMHL_ResultVector и в VMHL_Result не содержится решение задачи.

Пример значений рабочего вектора Parameters:

Parameters[0]=20;

Parameters[1]=100*100;

Принцип работы:

Принцип прост. Берутся случайно CountOfFitness решений независимо друг от друга. Выбирается лучшее. Всё.

О функции:

В простонародье алгоритм называют "методом научного тыка".

Алгоритм оптимизации. Ищет максимум функции пригодности FitnessFunction.

Решением является бинарная строка, то есть вектор, состоящий из 0 и 1.

Код 241. Оптимизируемая функция

```
double Func(int *x,int VMHL_N)
{
//Сумма всех элементов массива
return TMHL_SumVector(x,VMHL_N);
}
//---------------------------------------------------------------------------
```

Код 242. Пример использования

```
        int LengthBinarString=50;//Длина хромосомы
        int CountOfFitness=50*50;//Число вычислений функции пригодности

        int *ParametersOfBinaryMonteCarloAlgorithm;
        ParametersOfBinaryMonteCarloAlgorithm=new int[2];
        ParametersOfBinaryMonteCarloAlgorithm[0]=LengthBinarString;//Длина хромосомы
        ParametersOfBinaryMonteCarloAlgorithm[1]=CountOfFitness;//Число вычислений це
           левой функции

        int *Decision;//бинарное решение
        Decision=new int[LengthBinarString];
        double ValueOfFitnessFunction;//значение функции пригодности в точке Decision
        int VMHL_Success=0;//Успешен ли будет запуск сГА

        //Запуск алгоритма
        VMHL_Success=MHL_BinaryMonteCarloAlgorithm (
           ParametersOfBinaryMonteCarloAlgorithm,Func, Decision, &
           ValueOfFitnessFunction);

        //Используем полученный результат
        MHL_ShowNumber(VMHL_Success,"Как прошел запуск","VMHL_Success");
        //Как прошел запуск:
        //VMHL_Success=1

        if (VMHL_Success==1)
         {
         MHL_ShowVectorT(Decision,LengthBinarString,"Найденное решение","Decision");
         //Найденное решение:
         //Decision =
         //1   0  1  1  1  1  1  0  1  1  1  1  1  1  1  0  0  1  0  0  1  1  1  1  1
               0  1  1  0  1  1  0  1  1  1  1  0  0  1  1  1  1  0  1  1  1  0  1  1
               1

         MHL_ShowNumber(ValueOfFitnessFunction,"Значение функции пригодности","
            ValueOfFitnessFunction");
         // Значение функции пригодности:
         //ValueOfFitnessFunction=37
         }
        delete [] ParametersOfBinaryMonteCarloAlgorithm;
        delete [] Decision;
```

6.10 Перевод единиц измерений

6.10.1 MHL_DegToRad

Функция переводит угол из градусной меры в радианную.

Код 243. Синтаксис

```
double MHL_DegToRad(double VMHL_X);
```

Входные параметры:

VMHL_X — градусная мера угла.

Возвращаемое значение: Радианная мера угла.

Код 244. Пример использования

```
        double Rad;
        double Deg=90;//Угол в градусах

        //Вызов функции
        Rad=MHL_DegToRad(Deg);

        //Используем полученный результат
        MHL_ShowNumber(Rad,"Угол "+MHL_NumberToText(Deg)+" градусов","равен в радиана
            х");
        //Угол 90 градусов:
        //равен в радианах=1.5708
```

6.10.2 MHL_RadToDeg

Функция переводит угол из радианной меры в градусную.

Код 245. Синтаксис

```
double MHL_RadToDeg(double VMHL_X);
```

Входные параметры:

VMHL_X — радианная мера угла.

Возвращаемое значение: Градусная мера угла.

Код 246. Пример использования

```
        double Deg;
        double Rad=MHL_PI;//Угол в радианах

        //Вызов функции
        Deg=MHL_RadToDeg(Rad);

        //Используем полученный результат
        MHL_ShowNumber(Deg,"Угол "+MHL_NumberToText(Rad)+" радиан","равен в градусах"
            );
        //Угол 3.14159 радиан:
        //равен в градусах=180
```

6.11 Случайные объекты

6.11.1 MHL_BitNumber

Функция с вероятностью P (или 0.5 в переопределенной функции) возвращает 1. В противном случае возвращает 0.

Код 247. Синтаксис

```
int MHL_BitNumber(double P);
int MHL_BitNumber();
```

Есть две функции с разным набором аргументов.

Для первой функции:

Входные параметры:

P — вероятность появления 1.

Возвращаемое значение: 1 или 0.

Для второй функции:

Входные параметры:

Отсутствуют.

Возвращаемое значение: 1 или 0.

Код 248. Пример использования

```
        int x;
        double P=0.8;//Угол в радианах

        //Вызов функции
        x=MHL_BitNumber(P);

        //Используем полученный результат
        MHL_ShowNumber(x,"Из 0 и 1 с вероятностью "+MHL_NumberToText(P),"выбрано");

        //Вызов функции
        x=MHL_BitNumber();

        //Используем полученный результат
        MHL_ShowNumber(x,"Из 0 и 1 с вероятностью 0.5","выбрано");
```

6.11.2 MHL_RandomRealMatrix

Функция заполняет матрицу случайными вещественными числами из определенного интервала [Left;Right].

Код 249. Синтаксис

```
void MHL_RandomRealMatrix(double **VMHL_ResultMatrix, double Left, double Right, int
    VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу;

Left — левая граница интервала;

Right — правая граница интервала;

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение: Отсутствует.

Код 250. Пример использования

```
        int i;
        int VMHL_N=3;//Размер массива (число строк)
        int VMHL_M=3;//Размер массива (число столбцов)
        double **a;
        a=new double*[VMHL_N];
        for (i=0;i<VMHL_N;i++) a[i]=new double[VMHL_M];

        double Left=-3;//левая граница интервала;
        double Right=3;//правая граница интервала;

        //Вызов функции
        MHL_RandomRealMatrix(a,Left,Right,VMHL_N,VMHL_M);

        //Используем полученный результат
        MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
        //Случайная матрица:
        //a =
        //1.97571 0.862793 -0.357422
        //-2.62701   -0.202515   -2.79932
        //1.38794 1.35535  -2.29449

        for (i=0;i<VMHL_N;i++) delete [] a[i];
        delete [] a;
```

6.11.3 MHL_RandomRealMatrixInCols

Функция заполняет матрицу случайными вещественными числами из определенного интервала. При этом элементы каждого столбца изменяются в своих пределах.

Код 251. Синтаксис

```
void MHL_RandomRealMatrixInCols(double **VMHL_ResultMatrix, double *Left, double *
    Right, int VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу;

Left — левые границы интервала изменения элементов столбца (размер VMHL_M);

Right — правые границы интервала изменения элементов столбца (размер VMHL_M);

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение: Отсутствует.

Код 252. Пример использования

```
int i;
int VMHL_N=3;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
double **a;
a=new double*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new double[VMHL_M];
double *Left;
Left=new double[VMHL_M];
double *Right;
Right=new double[VMHL_M];

Left[0]=-5;//левая границы интервала изменения 1 столбца
Right[0]=-4; //правая граница интервала изменения 1 столбца

Left[1]=0;//левая границы интервала изменения 2 столбца
Right[1]=3; //правая граница интервала изменения 2 столбца

Left[2]=100;//левая границы интервала изменения 3 столбца
Right[2]=200; //правая граница интервала изменения 3 столбца

//Вызов функции
MHL_RandomRealMatrixInCols(a,Left,Right,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
//Случайная матрица:
//a =
//-4.20267   2.20367  148.468
//-4.42432   2.09418  138.654
//-4.07089   1.95831  140.198

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
delete [] Left;
delete [] Right;
```

6.11.4 MHL_RandomRealMatrixInElements

Функция заполняет матрицу случайными вещественными числами из определенного интервала. При этом каждый элемент изменяется в своих пределах.

Код 253. Синтаксис

```
void MHL_RandomRealMatrixInElements(double **VMHL_ResultMatrix, double **Left, double
    **Right, int VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу;

Left — левые границы интервала изменения каждого элемента (размер VMHL_N x VMHL_M);

Right — правые границы интервала изменения каждого элемента (размер VMHL_N x VMHL_M);

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение: Отсутствует.

Код 254. Пример использования

```
int i,j;
int VMHL_N=3;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
double **a;
a=new double*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new double[VMHL_M];
double **Left;
Left=new double*[VMHL_N];
for (i=0;i<VMHL_N;i++) Left[i]=new double[VMHL_M];
double **Right;
Right=new double*[VMHL_N];
for (i=0;i<VMHL_N;i++) Right[i]=new double[VMHL_M];

//Возьмем для примера границы интервала равными около номера ячейки в матрице
for (i=0;i<VMHL_N;i++)
 for (j=0;j<VMHL_M;j++)
  {
  Left[i][j]=i*VMHL_N+j-0.1;
  Right[i][j]=Left[i][j]+0.2;
  }

//Вызов функции
MHL_RandomRealMatrixInElements(a,Left,Right,VMHL_N,VMHL_M);

//Используем полученный результат

MHL_ShowMatrix (Left,VMHL_N,VMHL_M,"Матрица левых границ", "Left");
// Матрица левых границ:
//Left =
//-0.1 0.9   1.9
//2.9  3.9   4.9
//5.9  6.9   7.9

MHL_ShowMatrix (Right,VMHL_N,VMHL_M,"Матрица правых границ", "Right");
// Матрица правых границ:
//Right =
//0.1  1.1   2.1
//3.1  4.1   5.1
//6.1  7.1   8.1

MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
// Случайная матрица:
//a =
//0.0829529  1.04504  1.9892
//2.90126 3.92388  4.90221
//5.96102 6.90623  8.09661

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
for (i=0;i<VMHL_N;i++) delete [] Left[i];
delete [] Left;
for (i=0;i<VMHL_N;i++) delete [] Right[i];
delete [] Right;
```

6.11.5 MHL_RandomRealMatrixInRows

Функция заполняет матрицу случайными вещественными числами из определенного интервала. При этом элементы каждой строки изменяются в своих пределах.

Код 255. Синтаксис

```
void MHL_RandomRealMatrixInRows(double **VMHL_ResultMatrix, double *Left, double *
    Right, int VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу;

Left — левые границы интервала изменения элементов строки (размер VMHL_N);

Right — правые границы интервала изменения элементов строки (размер VMHL_N);

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение: Отсутствует.

Код 256. Пример использования

```
int i;
int VMHL_N=3;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
double **a;
a=new double*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new double[VMHL_M];
double *Left;
Left=new double[VMHL_N];
double *Right;
Right=new double[VMHL_N];

Left[0]=-5;//левая границы интервала изменения 1 строки
Right[0]=-4; //правая граница интервала изменения 1 строки

Left[1]=0;//левая границы интервала изменения 2 строки
Right[1]=3; //правая граница интервала изменения 2 строки

Left[2]=100;//левая границы интервала изменения 3 строки
Right[2]=200; //правая граница интервала изменения 3 строки

//Вызов функции
MHL_RandomRealMatrixInRows(a,Left,Right,VMHL_N,VMHL_M);

//Используем полученный результат

MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
// Случайная матрица:
//a =
//-4.98376   -4.64868 -4.38959
//1.14386 2.70071  2.76151
//141.309 192.12   100.122

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
delete [] Left;
```

```
        delete [] Right;
```

6.11.6 MHL_RandomRealVector

Функция заполняет массив случайными вещественными числами из определенного интервала [Left;Right].

Код 257. Синтаксис

```
void MHL_RandomRealVector(double *VMHL_ResultVector, double Left, double Right, int
    VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на массив;

Left — левая граница интервала;

Right — правая граница интервала;

VMHL_N — размер массива.

Возвращаемое значение: Отсутствует.

Код 258. Пример использования

```
        int VMHL_N=10;//Размер массива
        double *a;
        a=new double[VMHL_N];

        double Left=-3;
        double Right=3;

        //Вызов функции
        MHL_RandomRealVector(a,Left,Right,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Массив", "a");
        // Массив:
        //a =
        //1.73822
        //-0.406311
        //-2.7572
        //-0.351013
        //0.367493
        //1.40991
        //0.662476
        //-1.15576
        //-1.75781
        //-2.06927

        delete [] a;
```

6.11.7 MHL_RandomRealVectorInElements

Функция заполняет массив случайными вещественными числами из определенного интервала, где на каждую координату свои границы изменения.

Код 259. Синтаксис

```
void MHL_RandomRealVectorInElements(double *VMHL_ResultVector, double *Left, double *
    Right, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на массив;

Left — левые границы интервалов (размер VMHL_N);

Right — правые границы интервалов (размер VMHL_N)

VMHL_N — размер массива.

Возвращаемое значение: Отсутствует.

Код 260. Пример использования

```
int VMHL_N=2;//Размер массива
double *a;
a=new double[VMHL_N];

double *Left;
Left=new double[VMHL_N];
Left[0]=-3;//Левая граница изменения первого элемента массива
Left[1]=5;//Левая граница изменения второго элемента массива

double *Right;
Right=new double[VMHL_N];
Right[0]=3;//Правая граница изменения первого элемента массива
Right[1]=10;//Правая граница изменения второго элемента массива

//Вызов функции
MHL_RandomRealVectorInElements(a,Left,Right,VMHL_N);

//Используем полученный результат

MHL_ShowVector (Left,VMHL_N,"Массив левых границ", "Left");
// Массив левых границ:
//Left =
//-3
//5

MHL_ShowVector (Right,VMHL_N,"Массив правых границ", "Right");
// Массив правых границ:
//Right =
//3
//10

MHL_ShowVector (a,VMHL_N,"Случайных массив", "a");
// Случайных массив:
//a =
//1.32111
//6.5625
```

```
        delete [] a;
        delete [] Left;
        delete [] Right;
```

6.11.8 MHL_RandomVectorOfProbability

Функция заполняет вектор случайными значениями вероятностей. Сумма всех элементов вектора равна 1.

Код 261. Синтаксис

```
void MHL_RandomVectorOfProbability(double *VMHL_ResultVector, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на вектор вероятностей (одномерный массив);

VMHL_N — размер массива.

Возвращаемое значение: Отсутствует.

Код 262. Пример использования

```
        int VMHL_N=10;//Размер массива (число строк)
        double *a;
        a=new double[VMHL_N];

        //Заполним вектор случайными значениями вероятностей
        //Вызов функции
        MHL_RandomVectorOfProbability(a, VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Вектор вероятностей выбора", "a");
        // Вектор вероятностей выбора:
        //a =
        //0.0662721
        //0.0681826
        //0.083972
        //0.0554142
        //0.18878
        //0.160006
        //0.0698625
        //0.0652843
        //0.127822
        //0.114404

        MHL_ShowNumber (TMHL_SumVector(a,VMHL_N),"Его сумма", "Sum");
        // Его сумма:
        //Sum=1
```

6.11.9 TMHL_BernulliVector

Функция формирует случайный вектор Бернулли.

Код 263. Синтаксис

```
template <class T> void TMHL_BernulliVector(T *VMHL_ResultVector, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на вектор (одномерный массив);

VMHL_N — размер массива.

Возвращаемое значение:

Отсутствует.

Код 264. Пример использования

```
        int VMHL_N=10;//Размер массива (число строк)
        double *a;
        a=new double[VMHL_N];

        //Вызов функции
        TMHL_BernulliVector(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Случайный вектор Бернулли", "a");
      //Случайный вектор Бернулли:
        //a =
        //1
        //-1
        //1
        //1
        //-1
        //1
        //-1
        //-1
        //1
        //1
```

6.11.10 TMHL_RandomArrangingObjectsIntoBaskets

Функция предлагает случайный способ расставить N объектов в VMHL_N корзин при условии, что в каждой корзине может располагаться только один предмет.

Код 265. Синтаксис

```
template <class T> void TMHL_RandomArrangingObjectsIntoBaskets(T *VMHL_ResultVector,
    int N, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — массив, в который записывается результат;

N — число предметов;

VMHL_N — размер массива (и число корзин).

Возвращаемое значение:

Отсутствует.

Код 266. Пример использования

```
        int VMHL_N=10;//Размер массива
        int *a;
```

```
a=new int[VMHL_N];

int N=MHL_RandomUniformInt(0,10);// Размер турнира

//Вызов функции
TMHL_RandomArrangingObjectsIntoBaskets(a,N,VMHL_N);

//Используем полученный результат
MHL_ShowNumber (N,"Число предметов", "N");
// Число предметов:
// N=5
MHL_ShowVectorT (a,VMHL_N,"Случаное расположение по 10 корзинам", "a");
// Случаное расположение по 10 корзинам:
//a =
//0 1  0  0  0  1  1  0  1  1

delete [] a;
```

6.11.11 TMHL_RandomBinaryMatrix

Функция заполняет матрицу случайно нулями и единицами.

Код 267. Синтаксис

```
template <class T> void TMHL_RandomBinaryMatrix(T **VMHL_ResultMatrix,int VMHL_N,int
    VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на преобразуемый массив;

VMHL_N — размер массива VMHL_ResultMatrix (число строк);

VMHL_M — размер массива VMHL_ResultMatrix (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 268. Пример использования

```
int i;
int VMHL_N=10;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];

//Вызов функции
TMHL_RandomBinaryMatrix(a,VMHL_N,VMHL_M);

//Используем полученный результат
MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная бинарная матрица", "a");
//Случайная бинарная матрица:
//a =
//1 0  1
//0 0  0
//1 1  1
//1 0  0
```

```
//1 1  0
//1 1  0
//0 1  1
//0 0  1
//1 0  0
//1 1  0

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
```

6.11.12 TMHL_RandomBinaryVector

Функция заполняет вектор (одномерный массив) случайно нулями и единицами.

Код 269. Синтаксис

```
template <class T> void TMHL_RandomBinaryVector(T *VMHL_ResultVector,int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на преобразуемый массив;

VMHL_N — размер массива VMHL_ResultMatrix (число строк).

Возвращаемое значение:

Отсутствует.

Код 270. Пример использования

```
int VMHL_N=10;//Размер массива (число строк)
int *a;
a=new int[VMHL_N];

//Вызов функции
TMHL_RandomBinaryVector(a,VMHL_N);

//Используем полученный результат
MHL_ShowVector (a,VMHL_N,"Случайный бинарный вектор", "a");
//Случайный бинарный вектор:
//a =
//1
//1
//0
//0
//0
//0
//1
//1
//0
//0

delete [] a;
```

6.11.13 TMHL_RandomIntMatrix

Функция заполняет матрицу случайными целыми числами из определенного интервала [n;m).

Код 271. Синтаксис

```
template <class T> void TMHL_RandomIntMatrix(T **VMHL_ResultMatrix, T n, T m, int
    VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу;

n — левая граница интервала;

m — правая граница интервала;

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 272. Пример использования

```
        int i;
        int VMHL_N=3;//Размер массива (число строк)
        int VMHL_M=3;//Размер массива (число столбцов)
        int **a;
        a=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];

        int n=-3;//левая граница интервала;
        int m=3;//правая граница интервала;

        //Вызов функции
        TMHL_RandomIntMatrix(a,n,m,VMHL_N,VMHL_M);

        //Используем полученный результат

        MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
        // Случайная матрица:
        //a =
        //-1   -1 2
        //2 0  1
        //-3   2  -1ss

        for (i=0;i<VMHL_N;i++) delete [] a[i];
        delete [] a;
```

6.11.14 TMHL_RandomIntMatrixInCols

Функция заполняет матрицу случайными целыми числами из определенного интервала [n;m). При этом элементы каждого столбца изменяются в своих пределах.

Код 273. Синтаксис

```
template <class T> void TMHL_RandomIntMatrixInCols(T **VMHL_ResultMatrix, T *n, T *m,
    int VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу;

n — левые границы интервала изменения элементов столбцов (размер VMHL_M);

m — правые границы интервала изменения элементов столбцов (размер VMHL_M);

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 274. Пример использования

```
        int i;
        int VMHL_N=3;//Размер массива (число строк)
        int VMHL_M=3;//Размер массива (число столбцов)
        int **a;
        a=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
        int *n;
        n=new int[VMHL_M];
        int *m;
        m=new int[VMHL_M];

        n[0]=-50;//левая границы интервала изменения 1 столбца
        m[0]=-40; //правая граница интервала изменения 1 столбца

        n[1]=0;//левая границы интервала изменения 2 столбца
        m[1]=3; //правая граница интервала изменения 2 столбца

        n[2]=100;//левая границы интервала изменения 3 столбца
        m[2]=200; //правая граница интервала изменения 3 столбца

        //Вызов функции
        TMHL_RandomIntMatrixInCols(a,n,m,VMHL_N,VMHL_M);

        //Используем полученный результат

        MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
        //Случайная матрица:
        //a =
        //-47   2   142
        //-47   1   139
        //-44   0   199

        for (i=0;i<VMHL_N;i++) delete [] a[i];
        delete [] a;
        delete [] n;
        delete [] m;
```

6.11.15 TMHL_RandomIntMatrixInElements

Функция заполняет матрицу случайными целыми числами из определенного интервала [n;m). При этом каждый элемент изменяется в своих пределах.

Код 275. Синтаксис

```
template <class T> void TMHL_RandomIntMatrixInElements(T **VMHL_ResultMatrix, T **n,
    T **m, int VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу;

n — левые границы интервала изменения каждого элемента (размер VMHL_N x VMHL_M);

m — правые границы интервала изменения каждого элемента (размер VMHL_N x VMHL_M);

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 276. Пример использования

```
        int i,j;
        int VMHL_N=3;//Размер массива (число строк)
        int VMHL_M=3;//Размер массива (число столбцов)
        int **a;
        a=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
        int **n;
        n=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) n[i]=new int[VMHL_M];
        int **m;
        m=new int*[VMHL_N];
        for (i=0;i<VMHL_N;i++) m[i]=new int[VMHL_M];

        //Заполним границы изменения каждого элемента
        for (i=0;i<VMHL_N;i++)
         for (j=0;j<VMHL_M;j++)
          {
          n[i][j]=i*VMHL_N+j-10;
          m[i][j]=n[i][j]+20;
          }

        //Вызов функции
        TMHL_RandomIntMatrixInElements(a,n,m,VMHL_N,VMHL_M);

        //Используем полученный результат

        MHL_ShowMatrix (n,VMHL_N,VMHL_M,"Матрица левых границ", "n");
        //Матрица левых границ:
        //n =
        //-10  -9 -8
        //-7   -6 -5
        //-4   -3 -2
```

```
MHL_ShowMatrix (m,VMHL_N,VMHL_M,"Матрица правых границ", "m");
// Матрица правых границ:
//m =
//10   11 12
//13   14 15
//16   17 18

MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
// Случайная матрица:
//a =
//-4   6  -8
//-1   1  1
//-3   16 4

for (i=0;i<VMHL_N;i++) delete [] a[i];
delete [] a;
for (i=0;i<VMHL_N;i++) delete [] n[i];
delete [] n;
for (i=0;i<VMHL_N;i++) delete [] m[i];
delete [] m;
```

6.11.16 TMHL_RandomIntMatrixInRows

Функция заполняет матрицу случайными целыми числами из определенного интервала [n;m). При этом элементы каждой строки изменяются в своих пределах.

Код 277. Синтаксис

```
template <class T> void TMHL_RandomIntMatrixInRows(T **VMHL_ResultMatrix, T *n, T *m,
    int VMHL_N, int VMHL_M);
```

Входные параметры:

VMHL_ResultMatrix — указатель на матрицу;

n — левые границы интервала изменения элементов строки (размер VMHL_N);

m — правые границы интервала изменения элементов строки (размер VMHL_N);

VMHL_N — размер массива (число строк);

VMHL_M — размер массива (число столбцов).

Возвращаемое значение:

Отсутствует.

Код 278. Пример использования

```
int i;
int VMHL_N=3;//Размер массива (число строк)
int VMHL_M=3;//Размер массива (число столбцов)
int **a;
a=new int*[VMHL_N];
for (i=0;i<VMHL_N;i++) a[i]=new int[VMHL_M];
int *n;
n=new int[VMHL_N];
int *m;
```

```
        m=new int[VMHL_N];

        n[0]=-50;//левая границы интервала изменения 1 строки
        m[0]=-40; //правая граница интервала изменения 1 строки

        n[1]=0;//левая границы интервала изменения 2 строки
        m[1]=3; //правая граница интервала изменения 2 строки

        n[2]=100;//левая границы интервала изменения 3 строки
        m[2]=200; //правая граница интервала изменения 3 строки

        //Вызов функции
        TMHL_RandomIntMatrixInRows(a,n,m,VMHL_N,VMHL_M);

        //Используем полученный результат

        MHL_ShowMatrix (a,VMHL_N,VMHL_M,"Случайная матрица", "a");
        // Случайная матрица:
        //a =
        // -42 -42   -45
        //2 2  0
        //113  102   109

        for (i=0;i<VMHL_N;i++) delete [] a[i];
        delete [] a;
        delete [] n;
        delete [] m;
```

6.11.17 TMHL_RandomIntVector

Функция заполняет массив случайными целыми числами из определенного интервала [n,m).

Код 279. Синтаксис

```
template <class T> void TMHL_RandomIntVector(T *VMHL_ResultVector, T n, T m, int
    VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на массив;

n — левая граница интервала;

m — правая граница интервала;

VMHL_N — размер массива.

Возвращаемое значение:

Отсутствует.

Код 280. Пример использования

```
        int VMHL_N=10;//Размер массива
        int *a;
        a=new int[VMHL_N];

        int n=3;
        int m=50;
```

```
        //Вызов функции
        TMHL_RandomIntVector(a,n,m,VMHL_N);

        //Используем полученный результат

        MHL_ShowVector (a,VMHL_N,"Массив", "a");
        //Массив:
        //a =
        //6
        //23
        //40
        //19
        //39
        //37
        //48
        //46
        //31
        //42

        delete [] a;
```

6.11.18 TMHL_RandomIntVectorInElements

Функция заполняет массив случайными целыми числами из определенного интервала [n_i,m_i). При этом для каждого элемента массива свой интервал изменения.

Код 281. Синтаксис

```
template <class T> void TMHL_RandomIntVectorInElements(T *VMHL_ResultVector, T *n, T
    *m, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на массив;

n — указатель на массив левых границ интервала;

m — указатель на массив правых границ интервала;

VMHL_N — размер массива.

Возвращаемое значение:

Отсутствует.

Код 282. Пример использования

```
        int VMHL_N=2;//Размер массива
        int *a;
        a=new int[VMHL_N];

        int *n;
        n=new int[VMHL_N];
        n[0]=3;//Левая граница изменения первого элемента массива
        n[1]=-90;//Левая граница изменения второго элемента массива

        int *m;
        m=new int[VMHL_N];
```

```
m[0]=40;//Правая граница изменения первого элемента массива
m[1]=-10;//Правая граница изменения второго элемента массива

//Вызов функции
TMHL_RandomIntVectorInElements(a,n,m,VMHL_N);

//Используем полученный результат

MHL_ShowVector (n,VMHL_N,"Массив левых границ", "n");
//Массив левых границ:
//n =
//3
//-90

MHL_ShowVector (m,VMHL_N,"Массив правых границ", "m");
// Массив правых границ:
//m =
//40
//-10

MHL_ShowVector (a,VMHL_N,"Случайных массив", "a");
// Случайных массив:
//a =
//31
//-52

delete [] a;
delete [] n;
delete [] m;
```

6.12 Случайные числа

6.12.1 MHL_RandomNormal

Случайное число по нормальному закону распределения.

Код 283. Синтаксис

```
double MHL_RandomNormal(double Mean, double StdDev);
```

Входные параметры:

Mean — математическое ожидание;

StdDev — среднеквадратичное отклонение.

Возвращаемое значение: Случайное число по нормальному закону.

Код 284. Пример использования

```
double x;
double Mean=10;//математическое ожидание
double StdDev=3;//среднеквадратичное отклонение

//Вызов функции
x=MHL_RandomNormal(Mean,StdDev);

//Используем полученный результат
```

```
        MHL_ShowNumber(x,"Случайное число по нормальному закону (Mean="+
            MHL_NumberToText(Mean)+", StdDev="+MHL_NumberToText(StdDev)+")","x");
        //Случайное число по нормальному закону (Mean=10, StdDev=3):
        //x=10.9968
```

6.12.2 MHL_RandomUniform

Случайное вещественное число в интервале [a;b] по равномерному закону распределения.

Код 285. Синтаксис

```
double MHL_RandomUniform(double a, double b);
```

Входные параметры:

a — левая граница;

b — правая граница.

Возвращаемое значение: Случайное вещественное число в интервале [a;b].

Код 286. Пример использования

```
        double x;

        //Вызов функции
        x=MHL_RandomUniform(10,100);

        //Используем полученный результат
        MHL_ShowNumber(x,"Случайное число из интервала [10;100]","x");
        //Случайное числ
```

6.12.3 MHL_RandomUniformInt

Случайное целое число в интервале [n,m) по равномерному закону распределения.

Код 287. Синтаксис

```
int MHL_RandomUniformInt(int n, int m);
```

Входные параметры:

n — левая граница;

m — правая граница.

Возвращаемое значение: Случайное целое число от n до $m-1$ включительно.

Код 288. Пример использования

```
        double x;
        int s0=0,s1=0,s2=0,s3=0;

        //Вызов функции
        for (int i=0;i<1000;i++)
        {
        x=MHL_RandomUniformInt(0,3);
```

```
if (x==0) s0++;
if (x==1) s1++;
if (x==2) s2++;
if (x==3) s3++;
}

//Используем полученный результат
MHL_ShowNumber(x,"Случайное целое число из интервала [0;3)","x");
MHL_ShowNumber(s0,"Число выпадений 0","s0");
MHL_ShowNumber(s1,"Число выпадений 1","s0");
MHL_ShowNumber(s2,"Число выпадений 2","s0");
MHL_ShowNumber(s3,"Число выпадений 3","s0");
//Случайное целое число из интервала [0;3):
//x=1
//Число выпадений 0:
//s0=324
//Число выпадений 1:
//s0=374
//Число выпадений 2:
//s0=302
//Число выпадений 3:
//s0=0
```

6.13 Сортировка

6.13.1 TMHL_BubbleDescendingSort

Функция сортирует массив в порядке убывания методом "Сортировка пузырьком".

Код 289. Синтаксис

```
template <class T> void TMHL_BubbleDescendingSort(T *VMHL_ResultVector, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на исходный массив;

VMHL_N — количество элементов в массиве.

Возвращаемое значение:

Отсутствует.

Код 290. Пример использования

```
int i;
int VMHL_N=10;//Размер массива (число строк)
double *a;
a=new double[VMHL_N];
for (i=0;i<VMHL_N;i++)
 a[i]=MHL_RandomNumber();

MHL_ShowVector (a,VMHL_N,"Случайный вектор", "a");
// Например
// Случайный вектор:
//Случайный вектор:
//a =
//0.233978
```

```
        //0.29541
        //0.142914
        //0.719482
        //0.489319
        //0.610382
        //0.667908
        //0.596069
        //0.92099
        //0.88327

        //Вызов функции
        TMHL_BubbleDescendingSort(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Отсортированный вектор", "a");
        //Отсортированный вектор:
        //a =
        //0.92099
        //0.88327
        //0.719482
        //0.667908
        //0.610382
        //0.596069
        //0.489319
        //0.29541
        //0.233978
        //0.142914

        delete [] a;
```

6.13.2 TMHL_BubbleSort

Функция сортирует массив в порядке возрастания методом "Сортировка пузырьком".

Код 291. Синтаксис

```
template <class T> void TMHL_BubbleSort(T *VMHL_ResultVector, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на исходный массив;

VMHL_N — количество элементов в массиве.

Возвращаемое значение:

Отсутствует.

Код 292. Пример использования

```
        int i;
        int VMHL_N=10;//Размер массива (число строк)
        double *a;
        a=new double[VMHL_N];
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomNumber();

        MHL_ShowVector (a,VMHL_N,"Случайный вектор", "a");
        // Например
```

```
        //Случайный вектор:
        //a =
        //0.889862
        //0.575836
        //0.741882
        //0.0479736
        //0.788879
        //0.873413
        //0.343933
        //0.32196
        //0.0332031
        //0.0214844

        //Вызов функции
        TMHL_BubbleSort(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Отсортированный вектор", "a");
        // Отсортированный вектор:
        //a =
        //0.0214844
        //0.0332031
        //0.0479736
        //0.32196
        //0.343933
        //0.575836
        //0.741882
        //0.788879
        //0.873413
        //0.889862

        delete [] a;
```

6.13.3 TMHL_BubbleSortInGroups

Функция сортирует массив в порядке возрастания методом "Сортировка пузырьком"в группах данного массива. Имеется массив. Он делится на группы элементов по m элементов. Первые m элементов принадлежат первой группе, следующие m элементов - следующей и т.д. (Разумеется, в последней группе может и не оказаться m элементов). Потом в каждой группе элементы сортируются по возрастанию.

Код 293. Синтаксис

```
template <class T> void TMHL_BubbleSortInGroups(T *VMHL_ResultVector, int VMHL_N, int
    m);
```

Входные параметры:

VMHL_ResultVector — указатель на исходный массив;

VMHL_N — количество элементов в массиве;

m — количество элементов в группе.

Возвращаемое значение:

Отсутствует.

Код 294. Пример использования

```
int i;
int VMHL_N=9;//Размер массива (число строк)
double *a;
a=new double[VMHL_N];
for (i=0;i<VMHL_N;i++)
 a[i]=MHL_RandomUniformInt(10,50);

// Например
MHL_ShowVectorT (a,VMHL_N,"Случайный вектор", "a");
//Случайный вектор:
//a =
//20   42 39 19 27 33 35 44 32

int m=3;

//Вызов функции
TMHL_BubbleSortInGroups(a,VMHL_N,m);

//Используем полученный результат
MHL_ShowVectorT (a,VMHL_N,"Отсортированный вектор по три элемента", "a");
//Отсортированный вектор по три элемента:
//a =
//20   39 42 19 27 33 32 35 44

delete [] a;
```

6.13.4 TMHL_BubbleSortWithConjugateVector

Функция сортирует массив вместе с сопряженный массивом в порядке возрастания методом "Сортировка пузырьком". Пары элементов первого массива и сопряженного остаются без изменения.

Код 295. Синтаксис

```
template <class T, class T2> void TMHL_BubbleSortWithConjugateVector(T *
   VMHL_ResultVector, T2 *VMHL_ResultVector2, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на исходный массив;

VMHL_ResultVector2 — указатель на сопряженный массив;

VMHL_N — количество элементов в массиве.

Возвращаемое значение:

Отсутствует.

Код 296. Пример использования

```
int i;
int VMHL_N=10;//Размер массива (число строк)
double *a;
a=new double[VMHL_N];
int *b;
b=new int[VMHL_N];
```

```
for (i=0;i<VMHL_N;i++)
 {
 a[i]=MHL_RandomUniformInt(10,50);
 b[i]=MHL_RandomUniformInt(10,50);
 }

// Например
MHL_ShowVectorT (a,VMHL_N,"Случайный вектор", "a");
// Случайный вектор:
//a =
//31   32 13 26 40 40 47 26 10 18

MHL_ShowVectorT (b,VMHL_N,"Сопряженный вектор", "b");
//Сопряженный вектор:
//b =
//31   20 44 32 21 36 46 30 31 15

//Вызов функции
TMHL_BubbleSortWithConjugateVector(a,b,VMHL_N);

//Используем полученный результат
MHL_ShowVectorT (a,VMHL_N,"Отсортированный вектор", "a");
// Отсортированный вектор:
//a =
//10   13 18 26 26 31 32 40 40 47

MHL_ShowVectorT (b,VMHL_N,"Сопряженный вектор", "b");
// Сопряженный вектор:
//b =
//31   44 15 32 30 31 20 21 36 46

delete [] a;
delete [] b;
```

6.13.5 TMHL_BubbleSortWithTwoConjugateVectors

Функция сортирует массив вместе с двумя сопряженными массивами в порядке возрастания методом "Сортировка пузырьком". Пары элементов первого массива и сопряженного остаются без изменения.

Код 297. Синтаксис

```
template <class T, class T2, class T3> void TMHL_BubbleSortWithTwoConjugateVectors(T
    *VMHL_ResultVector, T2 *VMHL_ResultVector2, T3 *VMHL_ResultVector3, int VMHL_N);
```

Входные параметры:

VMHL_ResultVector — указатель на исходный массив;

VMHL_ResultVector2 — указатель на сопряженный массив;

VMHL_ResultVector3 — указатель на второй сопряженный массив;

VMHL_N — количество элементов в массивах.

Возвращаемое значение:

Отсутствует.

Код 298. Пример использования

```
int i;
int VMHL_N=10;//Размер массива (число строк)
double *a;
a=new double[VMHL_N];
int *b;
b=new int[VMHL_N];
int *c;
c=new int[VMHL_N];
for (i=0;i<VMHL_N;i++)
 {
 a[i]=MHL_RandomUniformInt(10,50);
 b[i]=MHL_RandomUniformInt(10,50);
 c[i]=MHL_RandomUniformInt(10,50);
 }

// Например
MHL_ShowVectorT (a,VMHL_N,"Случайный вектор", "a");
//Случайный вектор:
//a =
//45   27 11 18 24 25 16 19 34 43

MHL_ShowVectorT (b,VMHL_N,"Сопряженный вектор", "b");
//Сопряженный вектор:
//b =
//33   32 24 33 32 49 33 43 25 47

MHL_ShowVectorT (c,VMHL_N,"Сопряженный вектор", "c");
//Сопряженный вектор:
//c =
//15   24 27 43 17 47 25 11 13 26

//Вызов функции
TMHL_BubbleSortWithTwoConjugateVectors(a,b,c,VMHL_N);

//Используем полученный результат
MHL_ShowVectorT (a,VMHL_N,"Отсортированный вектор", "a");
//Отсортированный вектор:
//a =
//11   16 18 19 24 25 27 34 43 45

MHL_ShowVectorT (b,VMHL_N,"Сопряженный вектор", "b");
// Сопряженный вектор:
//b =
//24   33 33 43 32 49 32 25 47 33

MHL_ShowVectorT (c,VMHL_N,"Второй сопряженный вектор", "c");
//Второй сопряженный вектор:
//c =
//27   25 43 11 17 47 24 13 26 15

delete [] a;
delete [] b;
delete [] c;
```

6.14 Статистика и теория вероятности

6.14.1 MHL_DensityOfDistributionOfNormalDistribution

Плотность распределения вероятности нормированного и центрированного нормального распределения.

Код 299. Синтаксис

```
double MHL_DensityOfDistributionOfNormalDistribution(double x);
```

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Значение функции в точке.

Формула:

$$F(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}.$$

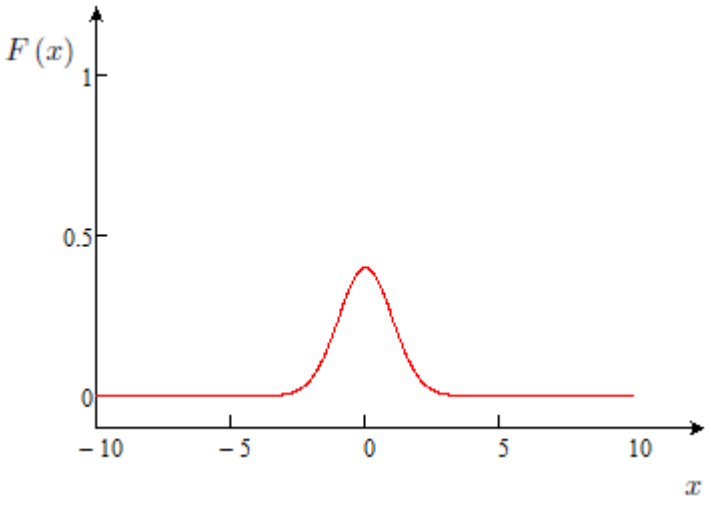

Рисунок 3. График функции

Код 300. Пример использования

```
double t;
double f;
t=MHL_RandomUniform(0,3);

//Вызов функции
f=MHL_DensityOfDistributionOfNormalDistribution(t);

//Используем полученный результат

MHL_ShowNumber (t,"Параметр", "t");
// Параметр:
//t=1.42401
MHL_ShowNumber (f,"Значение функции", "f");
// Значение функции:
```

```
        //f=0.144736
```

6.14.2 MHL_DistributionFunctionOfNormalDistribution

Функция распределения нормированного и центрированного нормального распределения.

Код 301. Синтаксис

```
double MHL_DistributionFunctionOfNormalDistribution(double x, double Epsilon);
```

Входные параметры:

x — входная переменная (правая граница интегрирования);

Epsilon — погрешность (например, Epsilon=0.001).

Возвращаемое значение:

Значение функции в точке.

Формула:

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{x^2}{2}}.$$

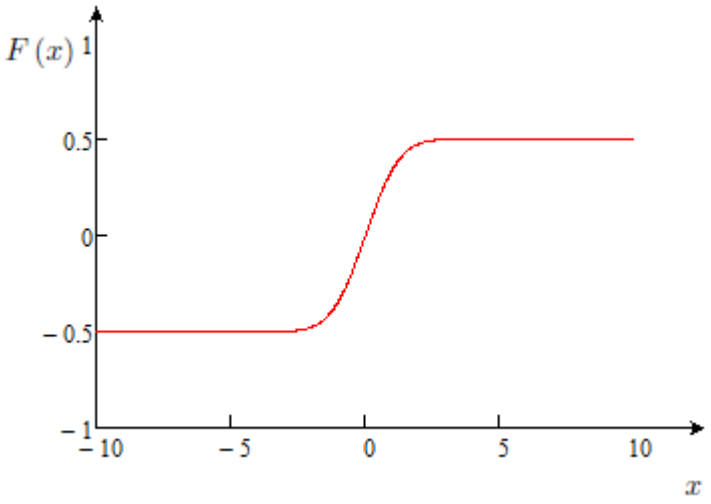

Рисунок 4. График функции

Код 302. Пример использования

```
        double t;
        double f;
        t=MHL_RandomUniform(0,3);

        //Вызов функции
        f=MHL_DistributionFunctionOfNormalDistribution(t,0.001);

        //Используем полученный результат

        MHL_ShowNumber (t,"Параметр", "t");
        //Параметр:
```

```
        //t=2.62253
        MHL_ShowNumber (f,"Значение функции", "f");
        //Значение функции:
        //f=0.495627
```

6.14.3 MHL_StdDevToVariance

Функция переводит среднеквадратичное уклонение в значение дисперсии случайной величины.

Код 303. Синтаксис

```
double MHL_StdDevToVariance(double StdDev);
```

Входные параметры:

StdDev — среднеквадратичное уклонение.

Возвращаемое значение:

Значение дисперсии случайной величины.

Код 304. Пример использования

```
        double Variance;
        double StdDev=6;

        //Вызов функции
        Variance=MHL_StdDevToVariance(StdDev);

        //Используем результат
        MHL_ShowNumber(Variance,"Дисперсия при среднеквадратичном уклонении, равным "
            +MHL_NumberToText(StdDev),"равна");
        //Дисперсия при среднеквадратичном уклонении, равным 6:
        //равна=2.44949
```

6.14.4 MHL_VarianceToStdDev

Функция переводит значение дисперсии случайной величины в среднеквадратичное уклонение.

Код 305. Синтаксис

```
double MHL_VarianceToStdDev(double Variance);
```

Входные параметры:

Variance — значение дисперсии случайной величины.

Возвращаемое значение:

Значение среднеквадратичного уклонения.

Код 306. Пример использования

```
        double StdDev;
```

```
double Variance=6;

//Вызов функции
StdDev=MHL_VarianceToStdDev(Variance);

//Используем полученный результат
MHL_ShowNumber(StdDev,"Среднеквадратичное уклонение при дисперсии, равной "+
    MHL_NumberToText(Variance),"равно");
//Среднеквадратичное уклонение при дисперсии, равной 6:
//равно=36
```

6.14.5 TMHL_Mean

Функция вычисляет среднее арифметическое массива.

Код 307. Синтаксис

```
template <class T> T TMHL_Mean(T *x, int VMHL_N);
```

Входные параметры:

x — массив;

VMHL_N — размер массива.

Возвращаемое значение:

Среднее арифметическое массива.

Код 308. Пример использования

```
int i;
int VMHL_N=10;//Размер массива
double *a;
a=new double[VMHL_N];
//Заполним случайными числами
for (i=0;i<VMHL_N;i++)
 a[i]=MHL_RandomUniform(0,10);

//Вызов функции
double Mean=TMHL_Mean(a,VMHL_N);

//Используем полученный результат
MHL_ShowVector (a,VMHL_N,"Массив", "a");
// Массив:
//a =
//4.65149
//4.00574
//1.41113
//1.55457
//2.75055
//3.16559
//8.26508
//3.86902
//9.5401
//4.50836

MHL_ShowNumber (Mean,"Среднее арифметическое массива", "Mean");
//Среднее арифметическое массива:
```

```
        //Mean=4.37216

        delete [] a;
```

6.14.6 TMHL_Median

Функция вычисляет медиану выборки.

Код 309. Синтаксис

```
template <class T> T TMHL_Median(T *x, int VMHL_N);
```

Входные параметры:

x — массив;

VMHL_N — размер массива.

Возвращаемое значение:

Медиана массива.

О функции:

Медиана (50-й процентиль, квантиль 0,5) — возможное значение признака, которое делит ранжированную совокупность (вариационный ряд выборки) на две равные части: 50

В случае, когда число элементов в выборке нечетно, то медиана равна элементу выборки посередине отсортированного массива.

В случае, когда число элементов в выборке четно, то медиана равна среднеарифметическому двух элементов выборки посередине отсортированного массива.

Код 310. Пример использования

```
        int i;
        int VMHL_N=MHL_RandomUniformInt(3,10);//Размер массива
        double *a;
        a=new double[VMHL_N];
        //Заполним случайными числами
        for (i=0;i<VMHL_N;i++)
         a[i]=MHL_RandomUniform(0,10);

        //Вызов функции
        double Median=TMHL_Median(a,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (a,VMHL_N,"Массив", "a");
        //Массив:
        //a =
        //8.77167
        //5.89142
        //6.45966
        //3.94775

        MHL_ShowNumber (Median,"Медиана", "Median");
        // Медиана:
        //Median=6.17554
```

```
        delete [] a;
```

6.14.7 TMHL_SampleCovariance

Функция вычисляет выборочную ковариацию выборки.

Код 311. Синтаксис

```
template <class T> T TMHL_SampleCovariance(T *x, T *y, int VMHL_N);
```

Входные параметры:

x — указатель на первую сравниваемую выборки;

y — указатель на вторую сравниваемую выборки;

VMHL_N — размер массивов.

Возвращаемое значение:

Значение выборочной ковариации.

Формула:

$$Cov\left(\bar{x},\bar{y}\right)=\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i-\frac{\sum_{j=1}^{n}x_j}{n}\right)\left(y_i-\frac{\sum_{j=1}^{n}y_j}{n}\right).$$

Код 312. Пример использования

```
        int VMHL_N=10;//Размер массива
        double *x;
        x=new double[VMHL_N];
        double *y;
        y=new double[VMHL_N];
        //Заполним случайными числами
        MHL_RandomRealVector (x,0,10,VMHL_N);
        MHL_RandomRealVector (y,0,10,VMHL_N);

        //Вызов функции
        double SampleCovariance=TMHL_SampleCovariance(x,y,VMHL_N);

        //Используем полученный результат
        MHL_ShowVector (x,VMHL_N,"Первый массив", "x");
        // Первый массив:
        //x =
        //3.06915
        //9.92218
        //2.5592
        //9.19586
        //8.23486
        //1.49231
        //3.93158
        //4.97345
        //6.78223
        //1.50909
```

6.14.8 TMHL_Variance

Функция вычисляет выборочную дисперсию выборки.

Код 313. Синтаксис

```
template <class T> T TMHL_Variance(T *x, int VMHL_N);
```

Входные параметры:

x — указатель на исходную выборку;

VMHL_N — размер массива.

Возвращаемое значение:

Выборочная дисперсия выборки.

Код 314. Пример использования

```
int VMHL_N=10;//Размер массива
double *x;
x=new double[VMHL_N];
//Заполним случайными числами
MHL_RandomRealVector (x,0,10,VMHL_N);

//Вызов функции
double Variance=TMHL_Variance(x,VMHL_N);

//Используем полученный результат
MHL_ShowVector (x,VMHL_N,"Массив", "x");
//Массив:
//x =
//4.61365
//6.74438
//0.18219
//9.68933
//8.77136
//2.5177
//1.89178
//6.16455
//8.45978
//4.33228

MHL_ShowNumber (Variance,"Значение выборочной дисперсии", "Variance");
//Значение выборочной дисперсии:
//Variance=10.1197

delete [] x;
```

6.15 Тригонометрические функции

6.15.1 MHL_Cos

Функция возвращает косинус угла в радианах.

Код 315. Синтаксис

```
double MHL_Cos(double x);
```

Входные параметры:

x — угол в радианах.

Возвращаемое значение:

Косинус угла.

Примечание:

Вводится только для того, чтобы множество тригонометрических функций было полным.

Код 316. Пример использования

```
        double y;
        double Angle=MHL_PI;//Угол в радинах

        //Вызов функции
        y=MHL_Cos(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Косинус угла "+MHL_NumberToText(Angle)+" радианов","равен")
            ;
        //Косинус угла 3.14159 радианов:
        //равен=-1
```

6.15.2 MHL_CosDeg

Функция возвращает косинус угла в градусах.

Код 317. Синтаксис

```
double MHL_CosDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Косинус угла.

Код 318. Пример использования

```
        double y;
        double Angle=180;//Угол в градусах

        //Вызов функции
        y=MHL_CosDeg(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Косинус угла "+MHL_NumberToText(Angle)+" градусов","равен")
            ;
        //Косинус угла 180 градусов:
        //равен=-1
```

6.15.3 MHL_Cosec

Функция возвращает косеканс угла в радианах.

Код 319. Синтаксис

```
double MHL_Cosec(double x);
```

Входные параметры:

x — угол в радианах.

Возвращаемое значение:

Косеканс угла.

Код 320. Пример использования

```
        double y;
        double Angle=MHL_PI/4.;//Угол в радинах

        //Вызов функции
        y=MHL_Cosec(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Косеканс угла "+MHL_NumberToText(Angle)+" радианов","равен"
            );
        //Косеканс угла 0.785398 радианов:
        //равен=1.41421
```

6.15.4 MHL_CosecDeg

Функция возвращает косеканс угла в градусах.

Код 321. Синтаксис

```
double MHL_CosecDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Косеканс угла.

Код 322. Пример использования

```
        double y;
        double Angle=45;//Угол в градусах

        //Вызов функции
        y=MHL_CosecDeg(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Косеканс угла "+MHL_NumberToText(Angle)+" градусов","равен"
            );
        //Косеканс угла 45 градусов:
        //равен=1.41421
```

6.15.5 MHL_Cotan

Функция возвращает котангенс угла в радианах.

Код 323. Синтаксис

```
double MHL_Cotan(double x);
```

Входные параметры:

x — угол в радианах.

Возвращаемое значение:

Котангенс угла.

Код 324. Пример использования

```
        double y;
        double Angle=MHL_PI/4.;//Угол в радинах

        //Вызов функции
        y=MHL_Cotan(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Котангенс угла "+MHL_NumberToText(Angle)+" радианов","равен
            ");
        //Котангенс угла 0.785398 радианов:
        //равен=1
```

6.15.6 MHL_CotanDeg

Функция возвращает котангенс угла в градусах.

Код 325. Синтаксис

```
double MHL_CotanDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Котангенс угла.

Код 326. Пример использования

```
        double y;
        double Angle=45;//Угол в градусах

        //Вызов функции
        y=MHL_CotanDeg(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Котангенс угла "+MHL_NumberToText(Angle)+" градусов","равен
            ");
        //Котангенс угла 45 градусов:
        //равен=1
```

6.15.7 MHL_Sec

Функция возвращает секанс угла в радианах.

Код 327. Синтаксис

```
double MHL_Sec(double x);
```

Входные параметры:

x — угол в радианах.

Возвращаемое значение:

Секанс угла.

Код 328. Пример использования

```
        double y;
        double Angle=MHL_PI/4.;//Угол в радинах

        //Вызов функции
        y=MHL_Sec(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Секанс угла "+MHL_NumberToText(Angle)+" радианов","равен");
        //Секанс угла 0.785398 радианов:
        //равен=1.41421
```

6.15.8 MHL_SecDeg

Функция возвращает секанс угла в градусах.

Код 329. Синтаксис

```
double MHL_SecDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Секанс угла.

Код 330. Пример использования

```
        double y;
        double Angle=45;//Угол в градусах

        //Вызов функции
        y=MHL_SecDeg(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Секанс угла "+MHL_NumberToText(Angle)+" градусов","равен");
        //Секанс угла 45 градусов:
        //равен=1.41421
```

6.15.9 MHL_Sin

Функция возвращает синус угла в радианах.

Код 331. Синтаксис

```
double MHL_Sin(double x);
```

Входные параметры:

x — угол в радианах.

Возвращаемое значение:

Синус угла.

Примечание:

Вводится только для того, чтобы множество тригонометрических функций было полным.

Код 332. Пример использования

```
        double y;
        double Angle=MHL_PI/2.;//Угол в радинах

        //Вызов функции
        y=MHL_Sin(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Синус угла "+MHL_NumberToText(Angle)+" радианов","равен");
        //Синус угла 1.5708 радианов:
        //равен=1
```

6.15.10 MHL_SinDeg

Функция возвращает синус угла в градусах.

Код 333. Синтаксис

```
double MHL_SinDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Синус угла.

Код 334. Пример использования

```
        double y;
        double Angle=90;//Угол в градусах

        //Вызов функции
        y=MHL_SinDeg(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Синус угла "+MHL_NumberToText(Angle)+" градусов","равен");
```

```
        //Синус угла 90 градусов:
        //равен=1
```

6.15.11 MHL_Tan

Функция возвращает тангенс угла в радианах.

Код 335. Синтаксис

```
double MHL_Tan(double x);
```

Входные параметры:

x — угол в радианах.

Возвращаемое значение:

Тангенс угла.

Примечание:

Вводится только для того, чтобы множество тригонометрических функций было полным.

Код 336. Пример использования

```
        double y;
        double Angle=MHL_PI/4.;//Угол в радинах

        //Вызов функции
        y=MHL_Tan(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Тангенс угла "+MHL_NumberToText(Angle)+" радианов","равен")
            ;
        //Тангенс угла 0.785398 радианов:
        //равен=1
```

6.15.12 MHL_TanDeg

Функция возвращает тангенс угла в градусах.

Код 337. Синтаксис

```
double MHL_TanDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Тангенс угла.

Код 338. Пример использования

```
        double y;
        double Angle=45;//Угол в градусах
```

```
        //Вызов функции
        y=MHL_TanDeg(Angle);

        //Используем полученный результат
        MHL_ShowNumber(y,"Тангенс угла "+MHL_NumberToText(Angle)+" градусов","равен")
            ;
        //Тангенс угла 45 градусов:
        //равен=1
```