MathHarrixLibrary v.3.1.0

А. Б. Сергиенко

22 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: **53** (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.

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. **TMHL_CheckElementInVector** — Функция проверяет наличие элемента a в векторе x.
2. **TMHL_EqualityOfVectors** — Функция проверяет равенство векторов.
3. **TMHL_FibonacciNumbersVector** — Функция заполняет массив числами Фибоначчи.
4. **TMHL_FillVector** — Функция заполняет вектор значениями, равных x.
5. **TMHL_MaximumOfVector** — Функция ищет максимальный элемент в векторе (одномерном массиве).
6. **TMHL_MinimumOfVector** — Функция ищет минимальный элемент в векторе (одномерном массиве).
7. **TMHL_NumberOfMaximumOfVector** — Функция ищет номер максимального элемента в векторе (одномерном массиве).
8. **TMHL_NumberOfMinimumOfVector** — Функция ищет номер минимального элемента в векторе (одномерном массиве).
9. **TMHL_NumberOfNegativeValues** — Функция подсчитывает число отрицательных значений в векторе (одномерном массиве).
10. **TMHL_NumberOfPositiveValues** — Функция подсчитывает число положительных значений в векторе (одномерном массиве).
11. **TMHL_NumberOfZeroValues** — Функция подсчитывает число нулевых значений в векторе (одномерном массиве).
12. **TMHL_OrdinalVector** — Функция заполняет вектор значениями, равные номеру элемента, начиная с единицы.
13. **TMHL_OrdinalVectorZero** — Функция заполняет вектор значениями, равные номеру элемента, начиная с нуля.
14. **TMHL_SearchFirstNotZero** — Функция возвращает номер первого ненулевого элемента массива.
15. **TMHL_SearchFirstZero** — Функция возвращает номер первого нулевого элемента массива.
16. **TMHL_SumSquareVector** — Функция вычисляет сумму квадратов элементов вектора.
17. **TMHL_SumVector** — Функция вычисляет сумму элементов вектора.
18. **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.
19. **TMHL_VectorMultiplyNumber** — Функция умножает вектор на число.
20. **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.

21. **TMHL_VectorToVector** — Функция копирует содержимое вектора (одномерного массива) в другой.
22. **TMHL_ZeroVector** — Функция зануляет массив.

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.

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.

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.

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 TMHL_CheckElementInVector

Функция проверяет наличие элемента a в векторе x.

Код 54. 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 55. 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;
```

6.1.2 TMHL_EqualityOfVectors

Функция проверяет равенство векторов.

Код 56. 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 57. 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;
```

6.1.3 TMHL_FibonacciNumbersVector

Функция заполняет массив числами Фибоначчи.

Код 58. 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 59. 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;
```

6.1.4 TMHL_FillVector

Функция заполняет вектор значениями, равных x.

Код 60. 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 61. 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;
```

6.1.5 TMHL_MaximumOfVector

Функция ищет максимальный элемент в векторе (одномерном массиве).

Код 62. 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 63. 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;
```

6.1.6 TMHL_MinimumOfVector

Функция ищет минимальный элемент в векторе (одномерном массиве).

Код 64. 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 65. 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;
```

6.1.7 TMHL_NumberOfMaximumOfVector

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

Код 66. 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 67. 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;
```

6.1.8 TMHL_NumberOfMinimumOfVector

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

Код 68. 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 69. 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;
```

6.1.9 TMHL_NumberOfNegativeValues

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

Код 70. 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 71. 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;
```

6.1.10 TMHL_NumberOfPositiveValues

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

Код 72. 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 73. 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;
```

6.1.11 TMHL_NumberOfZeroValues

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

Код 74. 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 75. 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;
```

6.1.12 TMHL_OrdinalVector

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

Код 76. 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 77. 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;
```

6.1.13 TMHL_OrdinalVectorZero

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

Код 78. 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 79. 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;
```

6.1.14 TMHL_SearchFirstNotZero

Функция возвращает номер первого ненулевого элемента массива.

Код 80. 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 81. 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;
```

6.1.15 TMHL_SearchFirstZero

Функция возвращает номер первого нулевого элемента массива.

Код 82. 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 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(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;
```

6.1.16 TMHL_SumSquareVector

Функция вычисляет сумму квадратов элементов вектора.

Код 84. 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 85. 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;
```

6.1.17 TMHL_SumVector

Функция вычисляет сумму элементов вектора.

Код 86. 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 87. 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;
```

6.1.18 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 88. 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 89. 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;
```

6.1.19 TMHL_VectorMultiplyNumber

Функция умножает вектор на число.

Код 90. 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 91. 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;
```

6.1.20 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 92. 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 93. 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;
```

6.1.21 TMHL_VectorToVector

Функция копирует содержимое вектора (одномерного массива) в другой.

Код 94. 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 95. 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;
```

6.1.22 TMHL_ZeroVector

Функция зануляет массив.

Код 96. 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 97. 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 Гиперболические функции

6.2.1 MHL_Cosech

Функция возвращает гиперболический косеканс.

Код 98. Синтаксис

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

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический косеканс.

Код 99. 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
```

6.2.2 MHL_Cosh

Функция возвращает гиперболический косинус.

Код 100. Синтаксис

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

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический косинус.

Код 101. 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
```

6.2.3 MHL_Cotanh

Функция возвращает гиперболический котангенс.

Код 102. Синтаксис

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

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический котангенс.

Код 103. 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
```

6.2.4 MHL_Sech

Функция возвращает гиперболический секанс.

Код 104. Синтаксис

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

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический секанс.

Код 105. 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
```

6.2.5 MHL_Sinh

Функция возвращает гиперболический синус.

Код 106. Синтаксис

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

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический синус.

Код 107. 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
```

6.2.6 MHL_Tanh

Функция возвращает гиперболический тангенс.

Код 108. Синтаксис

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

Входные параметры:

x — входная переменная.

Возвращаемое значение:

Гиперболический тангенс.

Код 109. 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
```

6.3 Перевод единиц измерений

6.3.1 MHL_DegToRad

Функция переводит угол из градусной меры в радианную.

Код 110. Синтаксис

```
double MHL_DegToRad(double VMHL_X);
```

Входные параметры:

VMHL_X — градусная мера угла.

Возвращаемое значение: Радианная мера угла.

Код 111. 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
            x");
        //Угол 90 градусов:
        //равен в радианах=1.5708
```

6.3.2 MHL_RadToDeg

Функция переводит угол из радианной меры в градусную.

Код 112. Синтаксис

```
double MHL_RadToDeg(double VMHL_X);
```

Входные параметры:

VMHL_X — радианная мера угла.

Возвращаемое значение: Градусная мера угла.

Код 113. 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
```

6.4 Случайные объекты

6.4.1 MHL_BitNumber

Функция с вероятностью P (или 0.5 в переопределенной функции) возвращает 1. В противном случае возвращает 0.

Код 114. 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 115. 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");
```

6.4.2 MHL_RandomRealMatrix

Функция заполняет матрицу случайными вещественными числами из определенного интервала [Left;Right].

Код 116. 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 117. 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;
```

6.4.3 MHL_RandomRealMatrixInCols

Функция заполняет матрицу случайными вещественными числами из определенного интервала. При этом элементы каждого столбца изменяются в своих пределах.

Код 118. 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 119. 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;
```

6.4.4 MHL_RandomRealMatrixInElements

Функция заполняет матрицу случайными вещественными числами из определенного интервала. При этом каждый элемент изменяется в своих пределах.

Код 120. 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 121. 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;
```

6.4.5 MHL_RandomRealMatrixInRows

Функция заполняет матрицу случайными вещественными числами из определенного интервала. При этом элементы каждой строки изменяются в своих пределах.

Код 122. 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 123. 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;
```

6.4.6 MHL_RandomRealVector

Функция заполняет массив случайными вещественными числами из определенного интервала [Left;Right].

Код 124. 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 125. 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;
```

6.4.7 MHL_RandomRealVectorInElements

Функция заполняет массив случайными вещественными числами из определенного интервала, где на каждую координату свои границы изменения.

Код 126. 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 127. 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;
```

6.4.8 MHL_RandomVectorOfProbability

Функция заполняет вектор случайными значениями вероятностей. Сумма всех элементов вектора равна 1.

Код 128. 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 129. 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
```

6.5 Случайные числа

6.5.1 MHL_RandomNormal

Случайное число по нормальному закону распределения.

Код 130. Синтаксис

```
double MHL_RandomNormal(double Mean, double StdDev);
```

Входные параметры:

Mean — математическое ожидание;

StdDev — среднеквадратичное отклонение.

Возвращаемое значение: Случайное число по нормальному закону.

Код 131. 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
```

6.5.2 MHL_RandomUniform

Случайное вещественное число в интервале [a;b] по равномерному закону распределения.

Код 132. 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 133. 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
```

6.5.3 MHL_RandomUniformInt

Случайное целое число в интервале [n,m) по равномерному закону распределения.

Код 134. 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 135. 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
```

6.6 Тригонометрические функции

6.6.1 MHL_Cos

Функция возвращает косинус угла в радианах.

Код 136. 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 137. 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
```

6.6.2 MHL_CosDeg

Функция возвращает косинус угла в градусах.

Код 138. Синтаксис

```
double MHL_CosDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Косинус угла.

Код 139. 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
```

6.6.3 MHL_Cosec

Функция возвращает косеканс угла в радианах.

Код 140. Синтаксис

```
double MHL_Cosec(double x);
```

Входные параметры:

x — угол в радианах.

Возвращаемое значение:

Косеканс угла.

Код 141. 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
```

6.6.4 MHL_CosecDeg

Функция возвращает косеканс угла в градусах.

Код 142. Синтаксис

```
double MHL_CosecDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Косеканс угла.

Код 143. 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
```

6.6.5 MHL_Cotan

Функция возвращает котангенс угла в радианах.

Код 144. Синтаксис

```
double MHL_Cotan(double x);
```

Входные параметры:

x — угол в радианах.

Возвращаемое значение:

Котангенс угла.

Код 145. 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
```

6.6.6 MHL_CotanDeg

Функция возвращает котангенс угла в градусах.

Код 146. Синтаксис

```
double MHL_CotanDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Котангенс угла.

Код 147. 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
```

6.6.7 MHL_Sec

Функция возвращает секанс угла в радианах.

Код 148. Синтаксис

```
double MHL_Sec(double x);
```

Входные параметры:

x — угол в радианах.

Возвращаемое значение:

Секанс угла.

Код 149. 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
```

6.6.8 MHL_SecDeg

Функция возвращает секанс угла в градусах.

Код 150. Синтаксис

```
double MHL_SecDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Секанс угла.

Код 151. 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
```

6.6.9 MHL_Sin

Функция возвращает синус угла в радианах.

Код 152. 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 153. 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
```

6.6.10 MHL_SinDeg

Функция возвращает синус угла в градусах.

Код 154. Синтаксис

```
double MHL_SinDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Синус угла.

Код 155. 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
```

6.6.11 MHL_Tan

Функция возвращает тангенс угла в радианах.

Код 156. 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 157. 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
```

6.6.12 MHL_TanDeg

Функция возвращает тангенс угла в градусах.

Код 158. Синтаксис

```
double MHL_TanDeg(double x);
```

Входные параметры:

x — угол в градусах.

Возвращаемое значение:

Тангенс угла.

Код 159. 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
```