MathHarrixLibrary v.3.0.0

А. Б. Сергиенко

19 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: **1**.

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 располагается 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 должны начинаться с **VMHL_**, чтобы различать их от внутренних.

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 реа-

лизацию 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.

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

5 Список функций

Вектора (Одномерные массивы)

1. **FuncF** — Умножает на 2.
2. **TMHL_FillVector** — Функция заполняет вектор значениями, равных x.

6 Список функций

Вектора (Одномерные массивы)

1. **TMHL_FillVector** — Функция заполняет вектор значениями, равных x.

7 Функции

7.1 Вектора (Одномерные массивы)

7.1.1 TMHL_FillVector

Функция заполняет вектор значениями, равных x.

Код 54. 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 55. 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;
```