# Лабораторная работа №7

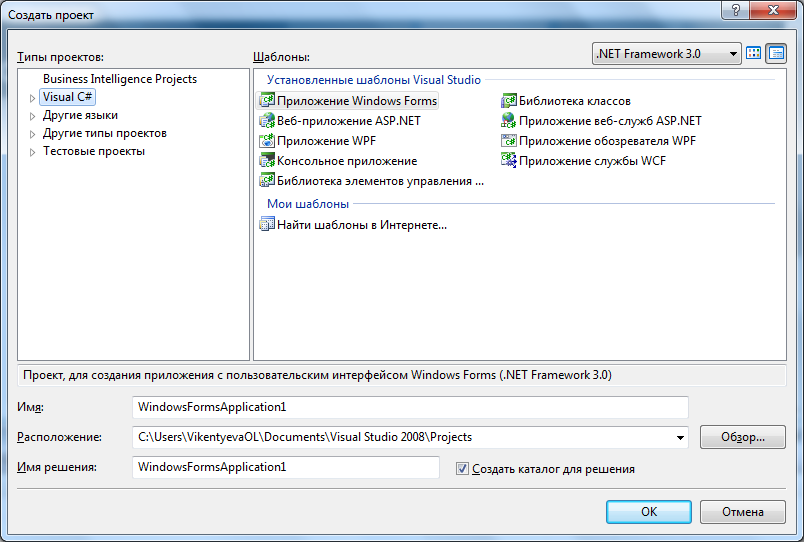
# Разработка Windows-приложения

## 1. Цель работы:

1. Получение практических навыков при работe с одномерными, многомерными и рваными массивами.
2. Получение практических навыков при работe с файлами.
3. Получение практических навыков при создании диалоговых Windows приложений.

## 2. Теоретические сведения

#### 2.1. Шаблон Windows - приложения



Среда создает заготовку формы и шаблон текста приложения.

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace Lection5\_4

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

{

static void Main()

Application.EnableVisualStyles();

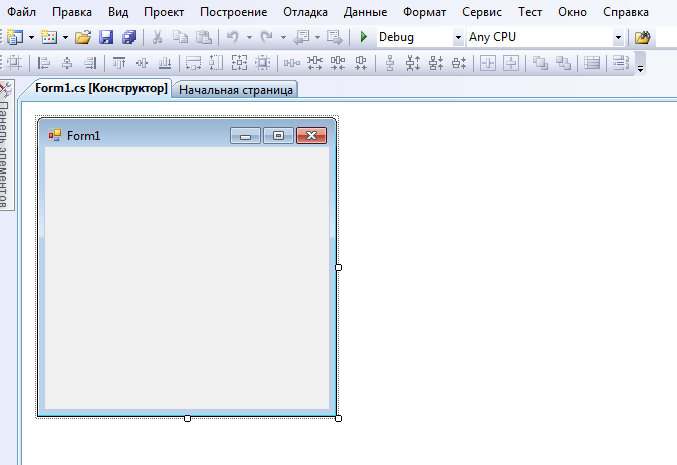
Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}



//Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Text;

using System.Windows.Forms;

namespace Lection5\_4

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

}

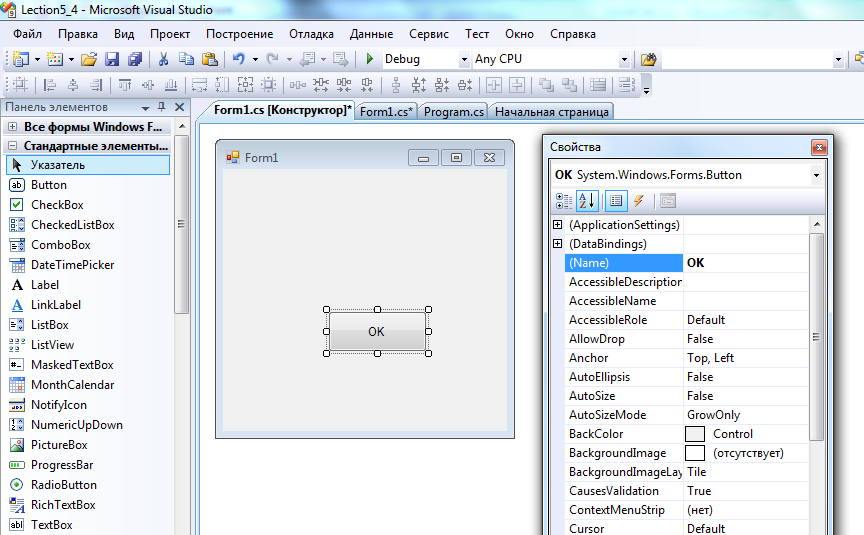
}

Процесс создания Windows-приложения состоит из двух основных этапов:

1. Визуальное проектирование, то есть задание внешнего облика приложения.

2. Определение поведения приложения путем написания процедур обработки событий.

Визуальное проектирование заключается в помещении на форму компонентов (элементов управления) и задании их свойств и свойств самой формы с помощью окна свойств.



События

Свойства

Определение поведения программы начинается с принятия решений, какие действия должны выполняться при щелчке на кнопках, вводе текста, выборе пунктов меню и т. д., иными словами, по каким событиям будут выполняться действия, реализующие функциональность программы.

Заготовка шаблона обработчика события формируется двойным щелчком на поле, расположенном справа от имени соответствующего события на вкладке Events окна свойств, при этом появляется вкладка окна редактора кода с заготовкой соответствующего обработчика.

namespace Lection5\_4

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void OK\_Click(object sender, EventArgs e)

{

}

}

}

#### 2.2.Основные типы Windows.Forms

|  |  |
| --- | --- |
| **Класс** | **Назначение** |
| Application | Класс Windows-приложения.  При помощи методов этого класса можно обрабатывать Windows-сообщения, запускать и прекращать работу приложения и т. п. |
| Button,  CheckBox,  ComboBox,  DataGrid,  GroupBox,  ListBox,  LinkLabel,  PictureBox. | Классы элементов управления (компоненты):  кнопка,  флажок,  комбинированный список,  таблица,  группа,  список,  метка с гиперссылкой,  изображение. |
| Form | Класс формы — окно Windows-приложения. |
| ColorDialog, FileDialog, FontDialog,  PrintPreviewDialog | Примеры стандартных диалоговых окон  для выбора цветов, файлов, шрифтов,  окно предварительного просмотра |
| Menu, MainMenu, Menultem, ContextMenu | Классы выпадающих и контекстных меню |
| Clipboard, Help, Timer, Screen, ToolTip,  Cursors | Вспомогательные типы для организации  графических интерфейсов: буфер обмена, помощь,  таймер, экран, подсказка, указатели мыши |
| StatusBar, Splitter, ToolBar, ScrollBar | Примеры дополнительных элементов управления,  размещаемых на форме: строка состояния,  разделитель, панель инструментов и т. д. |

Для каждого класса определен свой набор событий, на которые он может реагировать. Наиболее часто используемые события:

* Activated — получение формой фокуса ввода;
* Click, Doubleclick — одинарный и двойной щелчки мышью;
* Closed — закрытие формы;
* Load — загрузка формы;
* KeyDown, KeyUp — нажатие и отпускание любой клавиши и их сочетаний;
* Keypress — нажатие клавиши, имеющей ASCII-код;
* MouseDown, MouseUp — нажатие и отпускание кнопки мыши;
* MouseMove — перемещение мыши;
* Paint — возникает при необходимости прорисовки формы.

Класс Control является базовым для всех отображаемых элементов, то есть элементов, которые составляют графический интерфейс пользователя, например, кнопок, списков, полей ввода и форм. Класс Control реализует базовую функциональность интерфейсных элементов. Он содержит методы обработки ввода пользователя с помощью мыши и клавиатуры, определяет размер, положение, цвет фона и другие характеристики элемента.

При написании приложений применяются два способа обработки событий:

* замещение стандартного обработчика;
* задание собственного обработчика.

В большинстве случаев применяется второй способ. Среда разработки создает заготовку обработчика по двойному щелчку на поле, расположенном справа от имени соответствующего события на вкладке Events окна свойств. При этом в код приложения автоматически добавляется строка, регистрирующая этот обработчик.

#### 2.3. Элементы управления

Элементы управления, или компоненты, помещают на форму с помощью панели инструментов ToolBox . Обычно компоненты применяют в диалоговых окнах для получения данных от пользователя или его информирования.

Метка Label предназначена для размещения текста на форме. Текст хранится в свойстве Text. Метка не может получить фокус ввода, следовательно, не обрабатывает сообщения от клавиатуры.

Кнопка Button. Основное событие, обрабатываемое кнопкой, — щелчок мышью (Click). Кроме того, кнопка может реагировать на множество других событий — нажатие клавиш на клавиатуре и мыши, изменение параметров и т. д. Нажатие клавиши Enter или пробела, если при этом кнопка имеет фокус ввода, эквивалентно щелчку мышью на кнопке. Кнопки часто используются в диалоговых окнах. Как видно из названия, такое окно предназначено для диалога с пользователем и запрашивает у него какие-либо сведения (например, какой выбрать режим, или какой файл открыть). Диалоговое окно обладает свойством модальности. Это означает, что дальнейшие действия с приложением невозможны до того момента, пока это окно не будет закрыто. Закрыть окно можно, либо подтвердив введенную в него информацию щелчком на кнопке ОК (или Yes), либо отменив ее с помощью кнопки закрытия окна или, например, кнопки Cancel. Для сохранения информации о том, как было закрыто окно, у кнопки определяют свойство DialogResult. Это свойство может принимать стандартные значения из перечисления DialogResult, определенного в пространстве имен System. Windows. Forms.

Поле ввода TextBox позволяет пользователю вводить и редактировать текст, который запоминается в свойстве Text. Для обеспечения возможности ввода нескольких строк устанавливают свойства Multiline, Scrol1Bars и Wordwrap. Доступ только для чтения устанавливается с помощью свойства Readonly.

Элемент содержит методы очистки (Clear), выделения (Select), копирования в буфер (Сору), вставки из него (Paste) и др., а также реагирует на множество событий, основными из которых являются KeyPress и KeyDown.

Главное меню MainMenu размещается на форме таким же образом, как и другие компоненты: двойным щелчком на его значке на панели Toolbox. При этом значок располагается под заготовкой формы, а среда переходит в режим редактирования пунктов меню. Каждый пункт меню представляет собой объект типа Menultem, и при вводе пункта меню мы задаем его свойство Text. Переход к заданию заголовка следующего пункта меню выполняется либо щелчком мыши, либо нажатием клавиши Enter и клавиш со стрелками. Обычно, чтобы сделать программу понятнее, изменяют также свойства Name каждого пункта так, чтобы они соответствовали названиям пунктов.

Пункт меню может быть запрещен или разрешен (свойство Enabled), видим или невидим (Visible), отмечен или не отмечен (Checked). Заготовка обработчика событий формируется двойным щелчком на пункте меню.

Любое приложение обычно содержит в меню команду Exit, при выборе которой приложение завершается. Для закрытия приложения можно воспользоваться либо методом Close класса главной формы приложения, либо методом Exit класса Application,

Контекстное меню ContextMenu — это меню, которые вызывается во время выполнения программы по нажатию правой кнопки мыши на форме или элементе управления. Обычно в этом меню размещаются пункты, дублирующие пункты главного меню, или пункты, определяющие специфические для данного компонента действия. Контекстное меню создается и используется аналогично главному (значок контекстного меню появляется на панели инструментов, если воспользоваться кнопкой прокрутки). Для привязки контекстного меню к компоненту следует установить значение свойства ContextMenu этого компонента равным имени контекстного меню.

Флажок CheckBox используется для включения-выключения пользователем какого-либо режима. Для проверки, установлен ли флажок, анализируют его свойство Checked, принимающее значение true или false. Флажки используются в диалоговых окнах как поодиночке, так и в группе, причем все флажки устанавливаются независимо друг от друга.

Переключатель RadioButton позволяет пользователю выбрать один из нескольких предложенных вариантов, поэтому переключатели обычно объединяют в группы. Если один из них устанавливается (свойство Checked), остальные автоматически сбрасываются.

Программист может менять стиль и цвет текста, связанного с переключателем, и его выравнивание. Для переключателя можно задать цвет фона и фоновое изображение так же, как и для метки. Переключатели можно поместить непосредственно на форму, в этом случае все они составят одну группу. Если на форме требуется отобразить несколько групп переключателей, их размещают внутри компонента Group или Panel.

Панель GroupBox служит для группировки элементов на форме, например для того, чтобы дать общее название и визуально выделить несколько переключателей или флажков, обеспечивающих выбор связанных между собой режимов.

Список ListBox служит для представления перечня элементов, в которых пользователь может выбрать одно (свойство SelectionMode равно One) или несколько значений (свойство SelectionMode равно MultiSimple или MultiExtended). Если значение свойства SelectionMode равно MultiSimple, щелчок мышью на элементе выделяет его или снимает выделение. Значение Multi Extended позволяет использовать при выделении диапазона строк клавишу Shift, а при добавлении элемента — клавишу Ctrl, аналогично проводнику Windows. Запретить выделение можно, установив значение свойства SelectionMode, равное None. Чаще всего используются списки строк, но можно выводить и произвольные изображения.

Список может состоять из нескольких столбцов (свойство MultiColumn) и быть отсортированным в алфавитном порядке (Sorted = True). Элементы списка нумеруются с нуля. Они хранятся в свойстве Items, представляющем собой коллекцию. В Items можно добавлять элементы с помощью методовAddRange и Insert. Для удаления элементов служат методы Remove и RemoveAt, удаляющие заданный элемент и элемент по заданному индексу соответственно.

Выделенные элементы можно получить с помощью свойств Selectedltems и SelectedIndices, предоставляющих доступ к коллекциям выделенных элементов и их индексов.

#### 2.4. Диалоговые окна

Класс Form представляет собой заготовку формы, от которой наследуются классы форм приложения.

В библиотеке .NET нет специального класса для представления диалоговых окон. Вместо этого устанавливаются определенные значения свойств в классе обычной формы. В диалоговом окне можно располагать те же элементы управления, что и на обычной форме. Диалоговое окно характеризуется:

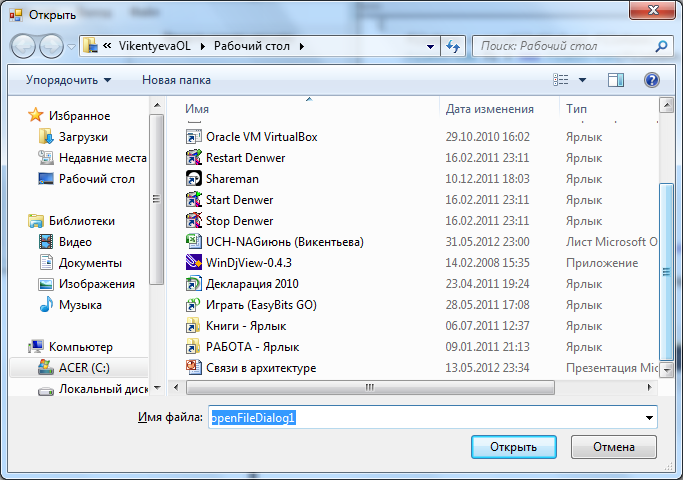
* неизменяемыми размерами (FormBorderStyle = FixedDialog);
* отсутствием кнопок восстановления и свертывания в правом верхнем углу заголовка формы (MaximizeBox = False, MinimizedBox = False);
* наличием кнопок наподобие ОК, подтверждающей введенную информацию, и Cancel, отменяющей ввод пользователя, при нажатии которых окно закрывается (AcceptButton = имя\_кнопки\_ОК, Cancel Button = имя\_кнопки\_ Cancel);
* установленным значением свойства DialogResult для кнопок, при нажатии которых окно закрывается.

Для отображения диалогового окна используется метод Show() или ShowModal(), который формирует результат выполнения из значений перечисления DialogResult. Если пользователь закрыл диалоговое окно щелчком на кнопке наподобие ОК, введенную им информацию можно использовать в дальнейшей работе. Закрытие окна щелчком на кнопке вроде Cancel отменяет все введенные данные. Диалоговое окно обычно появляется при выборе пользователем некоторой команды меню на главной форме.

#### 2.5. Стандартные диалоговые окна для работы с файлами

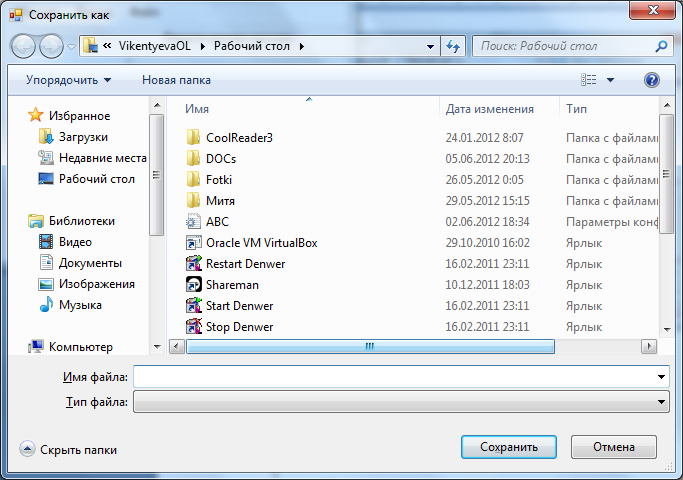
**OpenFileDialog (панель инструментов ToolBox).**

* Свойство FileName задает название файла, которое будет находиться в поле "Имя файла:" при появлении диалога.
* Свойство Filter задает ограничение файлов, которые могут быть выбраны для открытия. Через вертикальную разделительную линию можно задать смену типа расширения, отображаемого в выпадающем списке "Тип файлов" :(\*.txt)|\*.txt|All Files(\*.\*)|\*.\*, - обзор либо текстовых файлов, либо всех.
* Свойство InitialDirectory позволяет задать директорию, откуда будет начинаться обзор. Если свойство не установлено, исходной директорией будет рабочий стол.



**SaveFileDialog (панель инструментов ToolBox).**

Свойства этого диалога такие же, как у OpenFileDialog.



**2.6. Работа с файлами**

Для поддержки потоков библиотека .NET содержит иерархию классов.

**Основные классы пространства имен System.IO**

|  |  |
| --- | --- |
| Класс | Описание |
| BinaryReader,  BinaryWriter | Чтение и запись значений простых встроенных типов (целочисленных, логических, строковых и т. п.) во внутренней форме представления |
| FileStream | Произвольный (прямой) доступ к файлу, представленному как поток байтов |
| StreamWriter, StreamReader | Чтение из файла и запись в файл текстовой информации (произвольный доступ не поддерживается) |

Таким образом, выполнять обмен с внешними устройствами можно на уровне:

* двоичного представления данных (BinaryReader, BinaryWriter);
* байтов (FileStream);
* текста, то есть символов (StreamWriter, StreamReader).

Использование классов файловых потоков в программе предполагает следующие операции:

1. Создание потока и связывание его с физическим файлом.
2. Обмен (ввод-вывод).
3. Закрытие файла.

Каждый класс файловых потоков содержит несколько вариантов конструкторов, с помощью которых можно создавать объекты этих классов различными способами и в различных режимах. Например, файлы можно открывать только для чтения, только для записи или для чтения и записи.

**Режимы доступа к файлу содержатся в перечислении FileAccess, определенном в пространстве имен System.IO.**

|  |  |
| --- | --- |
| Значение | Описание |
| Read | Открыть файл только для чтения |
| ReadWrite | Открыть файл для чтения и записи |
| Write | Открыть файл только для записи |

**Возможные режимы открытия файла определены в перечислении FileMode**

|  |  |
| --- | --- |
| Значение | Описание |
| Append | Открыть файл, если он существует, и установить текущий указатель в конец файла. Если файл не существует, создать новый файл. |
| Create | Создать новый файл. Если в каталоге уже существует файл с таким же именем, он будет стерт. |
| CreateNew | Создать новый файл. Если в каталоге уже существует файл с таким же именем, возникает исключение IOException. |
| Open | Открыть существующий файл. |
| OpenOrCreate | Открыть файл, если он существует. Если нет, создать файл с таким именем |
| Truncate | Открыть существующий файл. После открытия он должен быть обрезан до нулевой длины. |

#### 2.2. Исключительные ситуации при работе с файлами

Операции по открытию файлов могут завершиться неудачно, например, при ошибке в имени существующего файла или при отсутствии свободного места на диске, поэтому рекомендуется всегда контролировать результаты этих операций.

В случае непредвиденных ситуаций среда выполнения генерирует различные исключения, обработку которых следует предусмотреть в программе, например:

* FileNotFoundException, если файла с указанным именем в указанном каталоге не существует;
* DirectoryNotFoundException, если не существует указанный каталог;
* Argument Except ion, если неверно задан режим открытия файла;
* IOException, если файл не открывается из-за ошибок ввода-вывода.

При закрытии файла освобождаются все связанные с ним ресурсы, например, для файла, открытого для записи, в файл выгружается содержимое буфера. Поэтому рекомендуется всегда закрывать файлы после окончания работы, в особенности файлы, открытые для записи. Если буфер требуется выгрузить, не закрывая файл, используется метод Flush.

#### 2.3. FileStream (Потоки байтов)

Ввод-вывод в файл на уровне байтов выполняется с помощью класса FileStream, который является наследником абстрактного класса Stream, определяющего набор стандартных операций с потоками.

**Элементы класса Stream**

|  |  |
| --- | --- |
| Значение | Описание |
| BeginRead,  BeginWrite | Начать асинхронный ввод или вывод |
| CanRead,  CanSeek,  CanWrite | Свойства, определяющие, какие операции поддерживает поток: чтение, прямой доступ и/или запись |
| Сlose | Закрыть текущий поток и освободить связанные с ним ресурсы (сокеты, указатели на файлы и т. п.) |
| EndRead,  EndWrite | Ожидать завершения асинхронного ввода; закончить асинхронный вывод |
| Flush | Записать данные из буфера в связанный с потоком источник данных и очистить буфер. Если для данного потока буфер не используется, то этот метод ничего не делает |
| Length | Возвратить длину потока в байтах |
| Position | Возвратить текущую позицию в потоке |
| Read,  ReadByte | Считать последовательность байтов (или один байт) из текущего потока и переместить указатель в потоке на количество считанных байтов |
| Seek | Установить текущий указатель потока на заданную позицию |
| SetLength | Установить длину текущего потока |
| Write,  WriteByte | Записать последовательность байтов (или один байт) в текущий потоки переместить указатель в потоке на количество записанных байтов |

Текущая позиция в потоке первоначально устанавливается на начало файла (для любого режима открытия, кроме Append) и сдвигается на одну позицию при записи каждого байта.

Для установки желаемой позиции чтения используется метод Seek, имеющий два параметра: первый задает смещение в байтах относительно точки отсчета, задаваемой вторым. Точки отсчета задаются константами перечисления SeekOrigin:

* начало файла — Begin,
* текущая позиция — Current,
* конец файла — End.

#### 2.4. StreamWriter и StreamReader (Потоки символов)

Символьные потоки StreamWriter и StreamReader работают с Unicode-символами, следовательно, ими удобнее всего пользоваться для работы с файлами, предназначенными для восприятия человеком. Эти потоки являются наследниками классов TextWriter и TextReader соответственно, которые обеспечивают их большей частью функциональности.

**КлассTextWriter**

|  |  |
| --- | --- |
| Значение | Описание |
| Close | Закрыть файл и освободить связанные с ним ресурсы. Если в процессе записи используется буфер, он будет автоматически очищен |
| Fl ush | Очистить все буферы для текущего файла и записать накопленные в них данные в место их постоянного хранения. Сам файл при этом не закрывается |
| NewLine | Используется для задания последовательности символов, означающих  начало новой строки. |
| Write | Записать фрагмент текста в поток |
| WriteLinе | Записать строку в поток и перейти на другую строку |

**Класс TextReader**

|  |  |
| --- | --- |
| Значение | Описание |
| Peek | Возвратить следующий символ, не изменяя позицию указателя в файле |
| Read | Считать данные из входного потока |
| ReadBlock | Считать из входного потока указанное пользователем количество символов и записать их в буфер, начиная с заданной позиции |
| ReadLine | Считать строку из текущего потока и возвратить ее как значение типа string. Пустая строка (null) означает конец файла (EOF) |
| ReadToEnd | Считать все символы до конца потока, начиная с текущей позиции, и возвратить считанные данные как одну строку типа string |

#### 2.5. Двоичные файлы (BinaryWriter, BinaryReader)

Двоичные файлы (BinaryWriter, BinaryReader) хранят данные в том же виде, в котором они представлены в оперативной памяти, то есть во внутренней форме представления. Двоичные файлы применяются не для просмотра их человеком, а для использования в программах.

Выходной поток BinaryWriter поддерживает произвольный доступ, то есть имеется возможность выполнять запись в произвольную позицию двоичного файла. Двоичный файл открывается на основе базового потока, в качестве которого чаще всего используется поток FileStream. Входной двоичный поток содержит перегруженные методы чтения для всех простых встроенных типов данных.

**Основные методы двоичных потоков.**

**Класс BinaryWriter**

|  |  |
| --- | --- |
| Значение | Описание |
| BaseStream | Базовый поток, с которым работает объект BinaryWriter |
| Close | Закрыть поток |
| Fl ush | Очистить все буферы для текущего файла и записать накопленные в них данные в место их постоянного хранения. Сам файл при этом не закрывается |
| Seek | Установить позицию в текущем потоке |
| Write | Записать фрагмент текста в поток |

**Класс BinaryReader**

|  |  |
| --- | --- |
| Значение | Описание |
| BaseStream | Базовый поток, с которым работает объект BinaryWriter |
| Close | Закрыть поток |
| Peek Char | Возвратить следующий символ, не изменяя позицию указателя в файле |
| Read | Считать данные из входного потока и сохранить в массиве, передаваемом как входной параметр |
| ReadХХХХ | Считать из входного потока данные определенного типа |

## 3. Постановка задачи

1. Разработать Windows – приложение, реализующее работу с одномерными, двумерными и рваными массивами.
2. Реализовать сохранение массивов в файле и загрузку ранее сохраненного массива из файла.

## 4. Варианты

|  |  |  |  |
| --- | --- | --- | --- |
| № варианта | Одномерный массив | Двумерный массив | Рваный массив |
| 1 | Удалить первый четный элемент | Добавить строку с заданным номером | Удалить самую длинную строку |
| 2 | Удалить первый отрицательный элемент | Добавить столбец с заданным номером | Удалить самую короткую строку |
| 3 | Удалить элемент с заданным ключом (значением) | Добавить строку в конец матрицы | Удалить все строки, в которых встречаются нули |
| 4 | Удалить элемент равный среднему арифметическому элементов массива | Добавить столбец в конец матрицы | Удалить все строки, в которых встречается заданное число K |
| 5 | Удалить элемент с заданным номером | Добавить строку в начало матрицы | Удалить К строк, начиная с номера N |
| 6 | Удалить N элементов, начиная с номера K | Добавить столбец в начало матрицы | Удалить строки начиная с номера K1 и заканчивая номером К2 включительно |
| 7 | Удалить все четные элементы | Добавить К строк в конец матрицы | Удалить первую строку, в которой встречаются нули |
| 8 | Удалить все элементы с четными индексами | Добавить К столбцов в конец матрицы | Удалить первую строку, в которой встречается заданное число K |
| 9 | Удалить все нечетные элементы | Добавить К строк в начало матрицы | Удалить строку с заданным номером |
| 10 | Удалить все элементы с нечетными индексами | Добавить К столбцов в начало матрицы | Удалить все строки с четными номерами |
| 11 | Добавить элемент в начало массива | Удалить строку с номером К | Добавить К строк, начиная с номера N |
| 12 | Добавить элемент в конец массива | Удалить столбец с номером К | Добавить К строк в конец массива |
| 13 | Добавить К элементов в начало массива | Удалить строки, начиная со строки К1 и до строки К2 включительно | Добавить строку с заданным номером |
| 14 | Добавить К элементов в конец массива | Удалить столбцы, начиная со столбца К1 и до столбца К2 | Добавить строку в начало массива |
| 15 | Добавить К элементов, начиная с номера N | Удалить все четные строки | Добавить строку в конец массива |
| 16 | Добавить после каждого отрицательного элемента его модуль | Удалить все четные столбцы | Добавить К строк, начиная с номера N |
| 17 | Добавить после каждого четного элемента элемент со значением 0 | Удалить все строки, в которых есть хотя бы один нулевой элемент | Добавить К строк в конец массива |
| 18 | Добавить по К элементов в начало и в конец массива | Удалить все столбцы, в которых есть хотя бы один нулевой элемент | Добавить строку с заданным номером |
| 19 | Добавить элемент с номером К | Удалить строку, в которой находится наибольший элемент матрицы | Добавить строку в начало массива |
| 20 | Удалить элемент с заданным номером | Добавить строки после каждой четной строки матрицы | Добавить строку в конец массива |
| 21 | Удалить N элементов, начиная с номера K | Добавить столбцы после каждого четного столбца матрицы | Добавить К строк, начиная с номера N |
| 22 | Удалить все четные элементы | Добавить К строк, начиная со строки с номером N | Добавить К строк в конец массива |
| 23 | Удалить все элементы с четными индексами | Добавить К столбцов, начиная со столбца с номером N | Добавить строку с заданным номером |
| 24 | Удалить все нечетные элементы | Добавить строку после строки, содержащей наибольший элемент | Добавить строку в начало массива |
| 25 | Удалить все элементы с нечетными индексами | Добавить столбец после столбца, содержащего наибольший элемент | Добавить строку в конец массива |

## 5. Методические указания

1. Для выбора действий использовать меню.
2. Предусмотреть 2 способа формирования массивов: вручную (ввод значений с клавиатуры) и с помощью датчика случайных чисел.
3. При удалении элементов (строк, столбцов) предусмотреть ошибочные ситуации, т. е. ситуации, в которых будет выполняться попытка удаления элемента (строки, столбца) из пустого массива или количество удаляемых элементов будет превышать количество имеющихся элементов (строк, столбцов). В этом случае должно быть выведено сообщение об ошибке.
4. При попытке вывода пустого массива должно выводиться сообщение о том, что массив пустой.
5. При сохранении массива в файл должен создаваться новый файл.
6. При загрузке массива из имеющегося файла необходимо реализовать обработку ситуации, когда выполняется попытка прочитать массив типа, отличного от того, который был сохранен (например, сохраняли одномерный массив из 10 элементов, а чтение выполняется в двумерный массив из 3 строк и 4 столбцов).

## 

## 6. Содержание отчета

1. Постановка задачи (общая и для конкретного варианта).
2. Анализ задачи.
3. Проектирование форм для взаимодействия с пользователем.
4. Код для реализации поставленных задач.
5. Тесты для каждой функции.
6. Тест для комплексной задачи (интеграция).