Практическая работа №5 (+6)
Функции, многомерные и рваные массивы, строки

1. Цель работы:

1) Получение практических навыков при работе с многомерными и рваными массивами.
2) Получение практических навыков при работе с классами Array и String.
3) Получение практических навыков при работе с функциями, передаче данных в функции различными способами, получение результатов из функций различными способами.
4) Получение практических навыков при создании диалоговых консольных приложений.

2. Теоретические сведения

2.1. Динамическая память

Все переменные, объявленные в программе, размещаются в одной непрерывной области памяти, которую называют сегментом данных. Такие переменные не меняют своего размера в ходе выполнения программы и называются статическими. Размера сегмента данных может быть недостаточно для размещения больших массивов информации. Выходом из этой ситуации является использование динамической памяти. **Динамическая память** – это память, выделяемая программе для ее работы за вычетом сегмента данных, стека, в котором размещаются локальные переменные подпрограмм и собственно тела программы.

Для создания динамических переменных используют операцию new:
int [] Arr=new int[10];

2.2. Многомерные массивы

Многомерным называется такой массив, который характеризуется двумя или более измерениями, а доступ к отдельному элементу осуществляется посредством двух или более индексов.

Простейший многомерный массив - двумерный. В двумерном массиве позиция любого элемента определяется двумя индексами. Если представить двумерный массив в виде таблицы данных, то один индекс означает строку, а второй — столбец.

Чтобы объявить двумерный массив целочисленных значений размером 10x20 с именем table, достаточно записать следующее:

```
int [ , ] table = new int[10, 20];
int [, ,] multidim = new int [ 4 , 10, 3]; //трехмерный массив
```

Чтобы получить доступ к элементу двумерного массива, необходимо указать номера всех индексов, разделив их запятой.

```
table[3,5]=10;
multidim[1,2,3]=100;
```

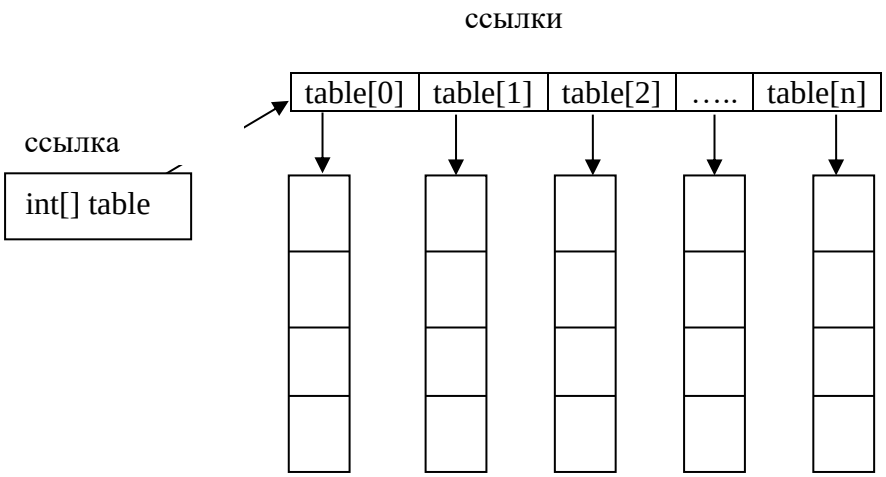


Рис. Выделение памяти под двумерный массив

Пример формирования двумерного массива:

```
Random rnd=new Random();
int strings, columns;
Console.WriteLine("Введите количество строк");
strings = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Введите количество столбцов");
columns = Convert.ToInt32(Console.ReadLine());
int[,] table = new int[strings, columns];
int i,j;
for (i = 0; i < strings; i++)
{
     for (j = 0; j < columns; j++)
      {
                    table[i, j] = rnd.Next(1, 10);
                    Console.Write(table[i, j] + "  ");
      }
      Console.WriteLine();
}
```

Многомерный массив можно инициализировать, заключив список инициализаторов каждой размерности в собственный набор фигурных скобок.

```
int[,] matr = {{11,12,13},
               {21,22,23}};

for (i = 0; i < 2; i++)
{
   for (j = 0; j < 3; j++)
      Console.Write(matr[i, j] + "  ");
      Console.WriteLine();
}
```

2.3. Рваные массивы

C# позволяет создавать двумерный массив у которого строки могут иметь различную длину (рваный массив). Следовательно, рваный массив можно использовать для создания таблицы со строками разной длины.

Рваные массивы объявляются с помощью наборов квадратных скобок, обозначающих размерности массива. Например, чтобы объявить двумерный рваный массив, используется следующий формат записи:

```
тип[][] имя = new тип[размер][];
```

Здесь элемент `размер` означает количество строк в массиве. Для самих строк память выделяется индивидуально, что позволяет строкам иметь разную длину.

```
Console.WriteLine("Формирование рваного массива");
Console.WriteLine("Введите количество строк");
strings = Convert.ToInt32(Console.ReadLine());
int[][] jag_arr = new int[strings][];
for (i = 0; i < strings; i++)
{
     Console.WriteLine("Введите количество столбцов");
     columns = Convert.ToInt32(Console.ReadLine());
     jag_arr[i] = new int[columns];
     for (j = 0; j < columns; j++)
          jag_arr[i][j] = rnd.Next(0, 10);
}
```

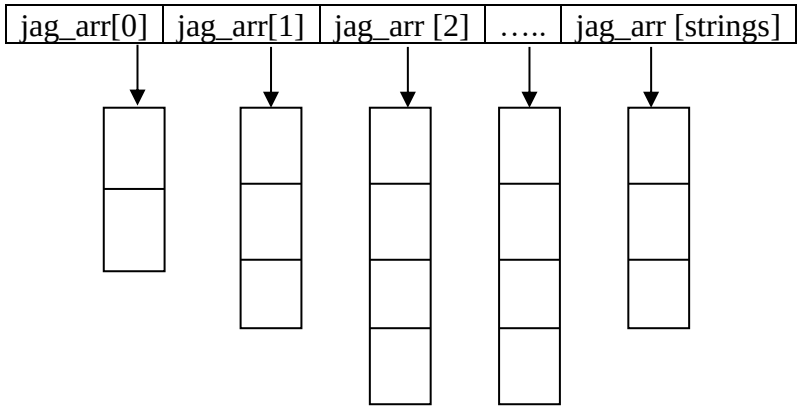


Рис. Выделение памяти под рваный массив

```
for (i = 0; i < strings; i++)
{
     for (j = 0; j < ragged_array[i].Length; j++)
     Console.Write(ragged_array[i][j] + "  ");
     Console.WriteLine();
 }
```

Инициализация рваного массива:

```
//1 способ
int [ ][ ] jagArr1=new int[3];//3строки
jagArr[0]=new int[]{1,2,3};
jagArr[1]=new int[]{1,2,3,4,5};
jagArr[2]=new int[]{1,2,3,4,5,6,7};
//2 способ
int [ ][ ]jagArr2=new int[3]//3строки
{
new int[ ]{1,2,3};
new int[ ]{1,2,3,4,5};
```

```
new int[ ]{1,2,3,4,5,6,7};
};
//3 способ
int [ ][ ]jagArr3
{
new int[ ]{1,2,3};
new int[ ]{1,2,3,4,5};
new int[ ]{1,2,3,4,5,6,7};
};
```

2.4. Использование цикла foreach для работы с многомерными и рваными массивами.

Цикл foreach последовательно опрашивает элементы массива в направлении от наименьшего индекса к наибольшему. При работе с многомерными массивами он возвращает элементы в порядке следования строк: от первой до последней.

```
//двумерный массив
int sum = 0;
foreach (int x in matr) sum += x;
Console.WriteLine("Сумма элементов массива равна " + sum);
//рваный массив
sum = 0;
for(i=0;i<strings;i++)
foreach (int x in ragged_array[i]) sum += x;
Console.WriteLine("Сумма элементов массива равна " + sum);
```

2.5. Базовый класс System. Array

Каждый создаваемый массив получает большую часть функциональности от класса System. Array.

Таблица 1. Функции класса System.Array

Clear ()	Статический метод, который позволяет устанавливать для всего ряда элементов в массиве пустые значения (0 — для чисел, null — для объектных ссылок и false — для булевских выражений)
CopyTo ()	Метод, который позволяет копировать элементы из исходного массива в целевой
Copy ()	Статический метод, который позволяет копировать заданный диапазон элементов одного массива в другой массив
Length	Свойство, которое возвращает информацию о количестве элементов в массиве
Rank	Свойство, которое возвращает информацию о количестве измерений в массиве
Reverse ()	Статическое свойство, которое представляет содержимое одномерного массива в обратном порядке
Sort ()	Статический метод, который позволяет сортировать одномерный массив
BinarySearch()	Статический метод, который находит номер элемента в упорядоченном одномерном массиве методом бинарного поиска
IndexOf()	Статический метод, который находит номер первого вхождения заданного элемента в одномерный массив
LastIndexOf()	Статический метод, который находит номер последнего вхождения заданного элемента в одномерный массив

2.6. Строковые и буквальные строковые литералы

Для представления текстовой информации в C# используются объекты класса string. Класс string представляет собой один из предопределенных типов языка C#. В .Net Framework этому типу соответствует класс System.String. Один из видов объектов класса

string мы уже многократно применяли — это строковые константы, или строковые литералы.

Строковая константа, или строковый литерал, имеет две формы:

- обычный (регулярный) строковый литерал (regular-string-literal);
- буквальный строковый литерал (verbatim-string-literal).

Регулярный строковый литерал - это последовательность символов и эскейп-последовательностей, заключенная в кавычки (не в апострофы).

Обрабатывая регулярный строковый литерал, компилятор из его символов формирует строковый объект и при этом заменяет эскейп-последовательности соответствующими кодами (символов или управляющих кодов). Например, литералу

"\u004F\x4E\u0045\ttwo"

будет соответствовать строка, при выводе которой на экране текст появится в таком виде:

ONE two

Буквальный (дословный) строковый литерал начинается с префикса @, за которым в кавычках размещается последовательность символов. Символы такого литерала воспринимаются буквально, т.е. в такой строке не обрабатываются эскейп-последовательности, а каждый символ воспринимается как таковой. В результате выполнения оператора:

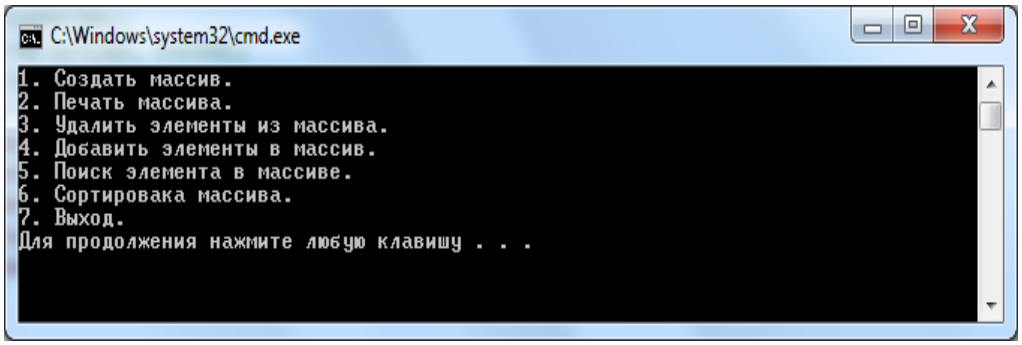
```
Console.WriteLine(@"\u004F\x4E\u0045\ttwo");
```

на экране появится \u004F\x4E\u0045\ttwo

Если в буквальном литерале необходимо поместить кавычку, то она изображается двумя рядом стоящими кавычками.

Буквальный литерал может быть размещен в коде программы на нескольких строках, и это размещение сохраняется при его выводе.

```
Console.WriteLine(@"1. Создать массив.
2. Печать массива.
3. Удалить элементы из массива.
4. Добавить элементы в массив.
5. Поиск элемента в массиве.
6. Сортировака массива.
7. Выход.");
```



2.7. Ссылки типа string

Каждый строковый литерал — это объект класса (типа) string.

```
string stroka;
```

Класс string является ссылочным типом. Кроме литералов, можно определить объекты класса string с использованием конструкторов. (Конструктор - специальный метод класса, предназначенный для инициализации объекта класса в процессе его создания.) Конструкторы класса string позволяют инициализировать объекты-строки несколькими способами.

```
string str1="Это строка 1";
```

```
char []charArr={'М','а','с','с','и','в'};
string str2=new string(charArr);

string str3=new string('S',5);

string str4 = new string(charArr, 4, 1);
```

Строковые объекты, как создаваемые с применением конструкторов, так и формируемые для представления строковых литералов, компилятор размещает в динамической памяти. Ссылки на строки размещаются в стеке. Размер строки при определении строкового объекта явно не указывается, он определяется автоматически при инициализации. **Ни размер строки, ни ее содержимое не могут изменяться после создания строки!!**

2.8. Операции над строками

Строки языка C# предназначены для хранения последовательностей символов, для каждого из которых отводится 2 байта, и они хранятся в кодировке Unicode (как данные типа char). В некотором смысле строка подобна одномерному массиву с элементами типа char. Элементы (символы строки) последовательно нумеруются, начиная с 0. Последний символ имеет номер на 1 меньше длины строки.

- **операция индексирования**:

`строка[индекс]`, индекс – целое число,>0.

Результат выражения с операцией индексирования - символ (значение типа **char),** размещенный в той позиции строки, номер которой соответствует индексному выражению. Если значение индекса меньше нуля, а также больше или равно длине строки, возникает исключительная ситуация (генерируется исключение).

- **Операция присваивания (=)** для строк выполняется не так как для массивов. Когда ссылке с типом массива присваивается значения ссылки на другой уже существующий массив, изменяет только значение ссылки. Массив, как объект, становится доступен для нескольких ссылок. Т.е. адрес первого элемента массива хранится в нескольких переменных ссылочного типа.
- Операция присваивания для строк приводит к созданию нового экземпляра той строки, на которую ссылается выражение справа от знака операции =. Ранее существовавшая строка никак не ассоциируется с новой ссылкой.
- **Операции сравнения** на равенство == и неравенство !=, применяемые к строкам, сравнивают последовательности символов в строках. (Для массивов сравниваются значения ссылок.)
- **Сцепление (конкатенацию) строк выполняет операция +.**

2.9. Методы и свойства класса String

int Length	свойство, позволяющее получить длину (количество символов) конкретной строки (объекта класса **string)**
int CompareTo()	метод, который сравнивает две строки и возвращает целочисленное значение. Для двух строк **S1,** S2 результат положительный, если S1>S2, отрицательный, если **S1<S2,** и нулевой, если SI == S2. Сравнение строк выполняется лексикографически.
static string Concat()	метод (их несколько) выполняет конкатенацию строк-параметров. Аргументов-строк может быть два, три или произвольное количество
static string Copy()	статический метод возвращает копию существующей строки.
int IndexOf()	нестатический метод поиска в вызывающей строке подстроки, заданной параметром. Возвращает индекс или -1, если поиск неудачен. Поиск - с начала строки.
string Insert()	нестатический метод для вставки строки-параметра в копию вызывающей строки с позиции, заданной дополнительным параметром
static string Join()	статический метод, объединяющий в одну строку строки массива-параметра. Первый параметр типа **string** задает разделитель, которым

	будут отделены друг от друга в результирующей строке элементы массива
int LastIndexOf()	нестатический метод поиска в вызывающей строке подстроки, заданной параметром. Возвращает индекс или -1, если поиск неудачен. Поиск с конца строки.
string Remove()	удаляет символы из копии строки
string Replace()	заменяет символы в копии строки
string [] Split()	формирует массив строк из фрагментов вызывающей строки. Параметр типа **char** задает разделители, которыми в строке разделены фрагменты
char [] ToCharArray()	копирует символы вызывающем строки в массив типа char[].
string Trim()	удаляет вхождение заданных символом (например, пробела) в начале и в конце строки
string Substring()	выделяет из строки подстроку. Параметры задают начало и длину выделяемой части строки.

2.6. Форматирование строк

При выводе, например, с помощью Console.Write(), значений базовых типов (например, int или double) они автоматически преобразуются в символьные строки. Если программиста не устраивает автоматически выбранный формат их внешнего представления, он может его изменить. Для этого можно воспользоваться статическим методом Format класса string или использовать так называемую строку форматирования в качестве первого параметра методов, поддерживающих форматирование, например, Console.Write() и Console.WriteLine (). В обоих случаях правила подготовки исходных данных для получения желаемого результата (новой строки) одинаковы.

```
static string Format (string form, params object[]ar);
```

- `string form` – строка форматирования, включает поля подстановок `{N[,W]:S[R]]},`

где N – номер аргумента,

W – ширина поля,

S – спецификатор формата,

R – спецификатор точности.

- `params object[]ar` – параметры, подставляемые вместо номера аргумента.

W - ширина поля в поле подстановки определяет количество позиций, выделяемых для изображения подставляемого значения. Если ширина поля не указана, то она определяется автоматически - минимально достаточной для изображения значения. Если ширина поля указана и превышает длину помещаемого в поле значения, то при положительной длине поля W значение выравнивается по правой границе. Если перед шириной поля W стоит минус, то выравнивание выполняется по левой границе поля.

Спецификатор формата S задает вид изображаемого значения.

C,c – валютный, R – количество десятичных разрядов.

D,d – целочисленный, R – минимальное количество цифр.

E,e – экспоненциальный, R – число разрядов после точки.

F,f – с фиксированной точкой, R – число разрядов после точки.

G,g – короткий из E или F.

X,x – шестнадцатеричный, R – минимальное число цифр.

2.7. Массивы строк

В массив помещаются не строки, а только ссылки на них, но при использовании массивов ссылок на строки не требуются никакие специальные операции для организации обращения к собственно строкам через ссылки на них. Поэтому в литературе, посвященной языку C#, зачастую говорят просто о массивах строк.

2.8. Неизменяемость объектов класса String

К символам объекта класса **string,** можно обращаться только для получения их значений. Например, для получения значения одного символа строки используется выражение с операцией индексирования []. Чтобы изменить строку можно воспользоваться следующим алгоритмом:

1. Переписать символы строки в массив с элементами типа char.
2. Выполнить преобразования в массиве с элементами типа char.
3. Создать новую строку, используя конструктор с параметром string(char[]).

2.9. Понятие функции

Функция – это именованная последовательность описаний и операторов, выполняющая законченное действие, например, формирование массива, печать массива и т. д.

Любая функция содержит одну или несколько инструкций. В хорошей программе одна функция выполняет только одну задачу. Каждая функция имеет имя, которое используется для ее вызова. В общем случае функции можно присвоить любое имя. Но имя Main () зарезервировано для функции, с которой начинается выполнение программы. Кроме того, в качестве имен функций нельзя использовать ключевые слова C#.

Рисунок 1 Функция как минимальный исполняемый модуль программы

Упрощенный формат записи функции следующий:

```
тип имя_функции([список_формальных параметров])
{
тело_функции
}
```

Тело_функции – это блок или составной оператор. Внутри функции нельзя определить другую функцию.

В теле функции должен быть оператор, который возвращает полученное значение функции в точку вызова. Он может иметь две формы:

1) `return выражение;`
2) `return;`

Первая форма используется для возврата результата, поэтому выражение должно иметь тот же тип, что и тип функции в определении. Вторая форма используется, если функция не возвращает значения, т. е. имеет тип `void`. Программист может не использовать этот оператор в теле функции явно, компилятор добавит его автоматически в конец функции перед `}`. Это может быть любой допустимый тип, включая типы классов, создаваемые программистом.

Список формальных параметров – это те величины, которые требуется передать в функцию. Элементы списка разделяются запятыми. Для каждого параметра указывается тип и имя. В объявлении имена можно не указывать.

Для того, чтобы выполнялись операторы, записанные в теле функции, функцию необходимо вызвать. При вызове указываются: имя функции и фактические параметры. Фактические параметры заменяют формальные параметры при выполнении операторов тела функции. Фактические и формальные параметры должны совпадать по количеству и типу.

Для вызова функции используется оператор

```
имя_метода (список аргументов);
```

В данной работе рассматриваются только методы классов, поэтому у каждого метода должен быть модификатор `static`.

Когда мы пишем свои методы (или используем уже готовые) важно понимать откуда метод берет свои данные и куда он будет отправлять результаты работы.

Исходные данные могут быть получены:

1) как параметры метода;
2) как глобальные переменные (по отношению к методу);
3) от внешних устройств (файлы, потоки ввода).

Результаты метод может передавать:

1) в точку вызова, как возвращаемое функцией значение;
2) в глобальные по отношению к методу объекты (переменные);
3) внешним устройствам (файлы, потоки вывода);
4) через параметры метода.

Глобальными объектами по отношению к методу являются все статические поля класса, в котором метод определен и статические поля других классов, к которым у метода есть доступ. Но обмен через глобальные объекты нарушает один из основополагающих принципов ООП – инкапсуляцию, поэтому в реальных разработках его использовать не рекомендуют (также как goto).

Обмен со стандартными потоками ввода-вывода реализуется с помощью класс Console.

2.10. Параметры функций

Основным способом обмена информацией между вызываемой и вызывающей функциями является механизм параметров.

При определении метода в его заголовке размещается спецификация параметров, она может быть пустой. Спецификация параметров – последовательность описаний параметров:

```
модификатор тип_параметра имя_параметра
```

Модификатор может отсутствовать или иметь одну из следующих форм: `ref`, `out`, `params`.

Параметр может быть любого типа (базового, строкой, object, массив, перечисление и т.д.). Имя параметра – идентификатор, выбираемый программистом. Область видимости и время действия параметра – заголовок и тело функции. Т.е. вне кода функции параметры не определены и недоступны.

Существуют следующие способы передачи параметров в функцию:

1. по значению;
2. по ссылке (`ref`);
3. выходные параметры (`out`);
4. массив-параметр (`params`).

При передаче по значению выполняются следующие действия:

1) вычисляются значения выражений, стоящие на месте фактических параметров;
2) в стеке выделяется память под формальные параметры функции;
3) каждому фактическому параметру присваивается значение формального параметра, при этом проверяются соответствия типов и при необходимости выполняются их преобразования.

Таким образом, при передаче параметров по значению в стек заносятся копии фактических параметров, и операторы функции работают с этими копиями. Доступа к самим фактическим параметрам у функции нет, следовательно, нет возможности их изменить.
Чтобы метод мог с помощью параметров изменять внешние по отношению к методу объекты, параметры должны иметь модификатор **ref**, т.е. передаваться **по ссылке**. Параметры, передаваемые по ссылке, используются для изменения уже существующих значений, внешних по отношению к методу объектов.
Выходные параметры снабжаются модификатором out и позволяют присвоить значения объектам вызывающего метода даже в тех случаях, когда эти объекты значений еще не имели. Локальным переменным, передаваемым в качестве выходных параметров, присваивать начальные значения не требуется (после вызова эти значения все равно будут утрачены). Причина, по которой компилятор позволяет передавать на первый взгляд неинициализированные данные, связана с тем, что в вызываемом методе операция присваивания должна выполняться обязательно.

После создания массив можно передавать как аргумент или получать в виде возвращаемого значения функции.

2.11. Функции с переменным числом параметров

В C# поддерживается использование массивов параметров за счет применения ключевого слова params. Ключевое слово params позволяет передавать методу переменное количество аргументов одного типа в виде единственного логического параметра. Аргументы, помеченные ключевым словом params, могут обрабатываться, если вызывающий код на их месте передает строго типизированный массив или разделенный запятыми список элементов.

2.12. Необязательные параметры

В определении функции может содержаться умалчиваемое значение параметра. Это значение используется, если при вызове функции соответствующий параметр опущен. Все параметры, описанные справа от такого параметра, также должны быть умалчиваемыми.

Значение, присваиваемое необязательному параметру, должно быть известно во время компиляции и не может вычисляться во время выполнения.

2.13. Именованные параметры

Именованные аргументы позволяют вызывать метод с указанием значений параметров в любом желаемом порядке. Следовательно, вместо того, чтобы передавать параметры исключительно в соответствии с позициями, в которых они определены (как приходится поступать в большинстве случаев), можно указывать имя каждого аргумента, двоеточие и конкретное значение. Именованные аргументы должны всегда размещаться в конце вызова метода.

2.14. Перегрузка методов

Цель перегрузки состоит в том, чтобы функция с одним именем по-разному выполнялась и возвращала разные значения при обращении к ней с различными типами и различным числом фактических параметров. Для обеспечения перегрузки необходимо для каждой перегруженной функции определить возвращаемые значения и передаваемые параметры так, чтобы каждая перегруженная функция отличалась от другой функции с тем же именем. Компилятор определяет, какую функцию выбрать по типу фактических параметров.

2.15. Использование меню для организации диалога с пользователем

Для организации взаимодействия с пользователем в консольных приложениях лучше всего использовать текстовое меню.

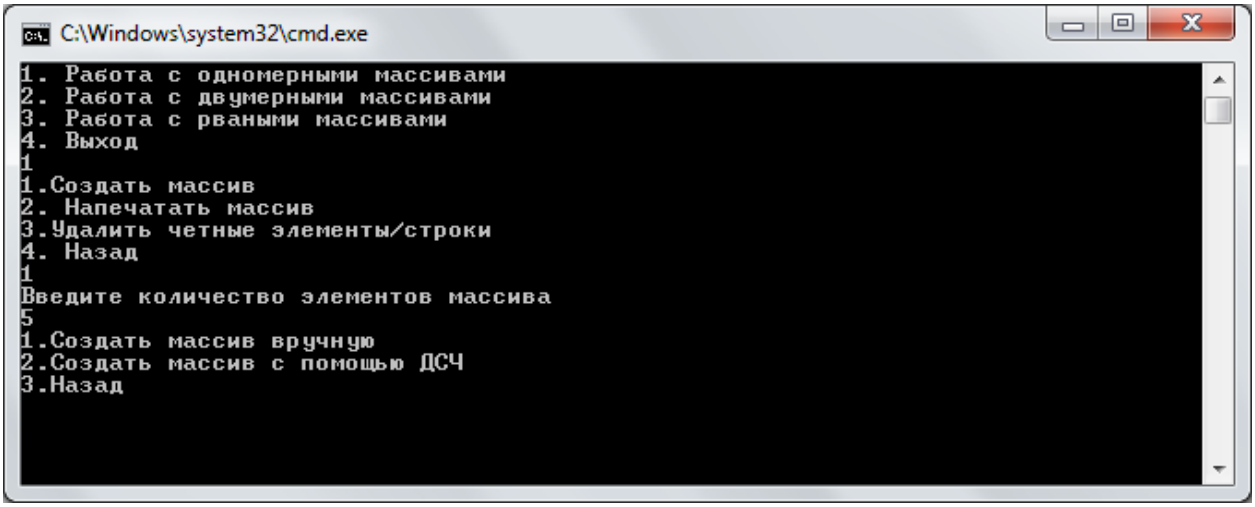


Рисунок 2. Пример меню

Для организации меню используется:

1) цикл с постусловием, в котором организуется печать пунктов меню и ввод выбранного пользователем пункта меню до тех пор, пока пользователь не выберет пункт «Выход»;

2) переключатель switch() для выполнения действий, реализующих выбранную пользователем операцию.

3. Постановка задачи

1. Сформировать динамический двумерный массив, заполнить его и вывести на печать.
2. Выполнить указанное в варианте задание и вывести полученный массив на печать.
3. Сформировать динамический рваный массив, заполнить его и вывести на печать.
4. Выполнить указанное в варианте задание и вывести полученный массив на печать.
5. Ввести строку символов. Строка состоит из слов, разделенных пробелами (пробелов может быть несколько) и знаками препинания (, ;:). В строке может быть несколько предложений, в конце каждого предложения стоит один знак препинания (.!?).
6. Выполнить обработку строки в соответствии с вариантом, используя по возможности, методы класса String.
7. Результаты обработки вывести на печать

5. Варианты

5.1. Массивы, класс Array

№ варианта	Двумерный массив	Рваный массив
1	Добавить строку с заданным номером	Удалить самую длинную строку
2	Добавить столбец с заданным номером	Удалить самую короткую строку

3	Добавить строку в конец матрицы	Удалить все строки, в которых встречаются нули
4	Добавить столбец в конец матрицы	Удалить все строки, в которых встречается заданное число K
5	Добавить строку в начало матрицы	Удалить K строк, начиная с номера N
6	Добавить столбец в начало матрицы	Удалить строки начиная с номера K1 и заканчивая номером K2 включительно
7	Добавить K строк в конец матрицы	Удалить первую строку, в которой встречаются нули
8	Добавить K столбцов в конец матрицы	Удалить первую строку, в которой встречается заданное число K
9	Добавить K строк в начало матрицы	Удалить строку с заданным номером
10	Добавить K столбцов в начало матрицы	Удалить все строки с четными номерами
11	Удалить строку с номером K	Добавить K строк, начиная с номера N
12	Удалить столбец с номером K	Добавить K строк в конец массива
13	Удалить строки, начиная со строки K1 и до строки K2 включительно	Добавить строку с заданным номером
14	Удалить столбцы, начиная со столбца K1 и до столбца K2	Добавить строку в начало массива
15	Удалить все четные строки	Добавить строку в конец массива
16	Удалить все четные столбцы	Добавить K строк, начиная с номера N
17	Удалить все строки, в которых есть хотя бы один нулевой элемент	Добавить K строк в конец массива
18	Удалить все столбцы, в которых есть хотя бы один нулевой элемент	Добавить строку с заданным номером
19	Удалить строку, в которой находится наибольший элемент матрицы	Добавить строку в начало массива
20	Добавить строки после каждой четной строки матрицы	Добавить строку в конец массива
21	Добавить столбцы после каждого четного столбца матрицы	Добавить K строк, начиная с номера N
22	Добавить K строк, начиная со строки с номером N	Добавить K строк в конец массива
23	Добавить K столбцов, начиная со столбца с номером N	Добавить строку с заданным номером
24	Добавить строку после строки, содержащей наибольший элемент	Добавить строку в начало массива
25	Добавить столбец после столбца, содержащего наибольший элемент	Добавить строку в конец массива

5.2. Строки, класс String

№	Задание	Пример выполнения
1	Перевернуть каждое нечетное предложение.	Исходная строка: В лесу родилась елочка. В лесу она росла. Зимой и летом стройная, зеленая была. Результат: Елочка родилась лесу в. В лесу она росла. Была зеленая, стройная летом и зимой.
2	Перевернуть каждое четное слово.	Исходная строка: В лесу родилась елочка. В лесу она росла. Зимой и летом стройная, зеленая была. Результат: В усел родилась акчоле. В усел она алсор. Зимой и летом яанйортс, зеленая алыб.
3	Определить есть ли в строке идентификаторы, если есть, то напечатать самый длинный идентификатор, если таких идентификаторов несколько, то напечатать все). Идентификаторы – имена объектов в программе (начинается с буквы или знака подчеркивания, включает только буквы и цифры).	Исходная строка: static void PrintUpper string info12346: WriteLine ToUpper info, 1234info. Результат: PrintUpper, info12346
4	Поменять местами первое и последнее предложение в строке.	Исходная строка: В лесу родилась елочка. В лесу она росла. Зимой и летом стройная, зеленая была. Результат: Зимой и летом стройная, зеленая была. В лесу она росла. В лесу родилась елочка.
5	Поменять местами первое и последнее слово в строке.	Исходная строка: В лесу родилась елочка. В лесу она росла. Зимой и летом стройная, зеленая была. Результат: Была лесу родилась елочка. В лесу она росла. Зимой и летом стройная, зеленая была в.
6	Определить есть ли в строке ключевые слова C#. Если есть, то напечатать сколько раз встречается каждое слово.	Исходная строка: static void PrintUpper string info12346: WriteLine ToUpper info, 1234info. if x>0 then sign=1; else if x<0 sign=-1; else sign=0. Результат: static – 1 void – 1 if – 2 else – 2
7	Сдвинуть циклически влево каждое слово на количество символов равное номеру этого слова в предложении.	Исходная строка: В лесу родилась елочка. В лесу она росла. Зимой и летом стройная, зеленая была. Результат: В суле иласьрод каелоч. В суле она аросл. Имойз и омлет йнаястро, аязелен лабы.
8	Перевернуть каждое предложение, заканчивающееся символом ’!’.	Исходная строка: В лесу родилась елочка! В лесу она росла. Зимой и летом стройная, зеленая была! Результат: Елочка родилась лесу в! В лесу она росла. Была зеленая, стройная летом и зимой!
9	Перевернуть каждое слово, номер которого в предложении, совпадает с его длиной.	Исходная строка: В лесу родилась елка! В лесу она росла. Зимой и летом была стройная, зеленая! Исходная строка: В лесу родилась **акле**! В лесу **ано** росла. Зимой и летом **алыб** стройная, зеленая!

10	Определить есть ли в строке идентификаторы, если есть, то напечатать самый короткий идентификатор. Идентификаторы – имена объектов в программе (начинается с буквы или знака подчеркивания, включает только буквы и цифры).	Исходная строка: static void PrintUpper string info12346: WriteLine ToUpper info, 1234info. Результат: info
11	Удалить первое и последнее предложение в строке.	Исходная строка: В лесу родилась елочка! В лесу она росла. Зимой и летом стройная, зеленая была! Результат: В лесу она росла
12	Удалить первое и последнее слово в каждом предложении.	Исходная строка: В лесу родилась елочка! В лесу она росла. Зимой и летом стройная, зеленая была! Результат: Лесу родилась! Лесу она. И летом стройная, зеленая!
13	Перевернуть все слова в каждом предложении и отсортировать слова по убыванию в лексикографическом порядке.	Исходная строка: В лесу родилась елочка! В лесу она росла. Результат: Акчоле в усел ьсалидор! Алсор ано в усел.
14	Перевернуть все слова в предложении и отсортировать слова по убыванию длин слов.	Исходная строка: В лесу родилась елочка! В лесу она росла. Результат: В усел акчоле ьсалидор! В ано усел алсор.
15	Удалить из строки все слова, которые начинаются и заканчиваются на один и тот же символ.	Исходная строка: В траве сидел кузнечик! Кузнечик не трогал козявок и дружил с мухом. Результат: В траве сидел! Не трогал и дружил с.
16	Определить есть ли в строке ключевые слова C#. Если есть, то напечатать сколько раз встречается каждое слово.	Исходная строка: static void PrintUpper string info12346: WriteLine ToUpper info, 1234info. if x>0 then sign=1; else if x<0 sign=-1; else sign=0. Результат: static – 1 void – 1 if – 2 else – 2
17	Сдвинуть циклически влево каждое слово на количество символов равное номеру этого слова в строке.	Исходная строка: В лесу родилась елочка. В лесу она росла. Зимой и летом стройная, зеленая была. Результат: В суле иласьрод каелоч. В суле она аросл. Имойз и омлет йнаястро, аязелен лабы.
18	Перевернуть каждое предложение, заканчивающееся символом ’!’.	Исходная строка: В лесу родилась елочка! В лесу она росла. Зимой и летом стройная, зеленая была! Результат: Елочка родилась лесу в! В лесу она росла. Была зеленая, стройная летом и зимой!
19	Перевернуть каждое слово, номер которого совпадает с его длиной.	Исходная строка: В лесу родилась елка! В лесу она росла. Зимой и летом была стройная, зеленая! Исходная строка: В лесу родилась **акле**! В лесу **ано** росла. Зимой и летом **алыб** стройная, зеленая!
20	Определить есть ли в строке идентификаторы, если есть, то напечатать самый короткий идентификатор.	Исходная строка: static void PrintUpper string info12346: WriteLine ToUpper info, 1234info. Результат: info

21	Удалить все предложения в строке, которые заканчиваются «!».	Исходная строка: В лесу родилась елочка! В лесу она росла. Зимой и летом стройная, зеленая была! Результат: В лесу она росла.
22	Удалить все слова в строке, которые начинаются с цифры .	Исходная строка: static void PrintUpper string info 12346: WriteLine ToUpper info, 1234info. Результат: static void PrintUpper string info: WriteLine ToUpper info.
23	Перевернуть все слова в предложении и отсортировать слова по убыванию в лексикографическом порядке.	Исходная строка: В лесу родилась елочка! В лесу она росла. Результат: Акчоле в усел ьсалидор! Алсор ано в усел.
24	Перевернуть все слова в предложении и отсортировать слова по убыванию длин слов.	Исходная строка: В лесу родилась елочка! В лесу она росла. Результат: В усел акчоле ьсалидор! В ано усел алсор.
25	Удалить из строки все слова, которые начинаются и заканчиваются на один и тот же символ.	Исходная строка: В траве сидел кузнечик! Кузнечик не трогал козявок и дружил с мухом. Результат: В траве сидел! Не трогал и дружил с.

5. Методические указания

1) Для организации взаимодействия с пользователем использовать текстовое меню.

2) Предусмотреть 2 способа формирования массивов: вручную (ввод значений с клавиатуры) и с помощью датчика случайных чисел.

3) Предусмотреть 2 способа ввода строк: с клавиатуры и из заранее сформированного массива строк (тестовый массив строк).

4) Предусмотреть возникновение исключительных ситуаций, например, при вводе символов вместо цифр числа.

5) 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.

6) При попытке вывода пустого массива или пустой строки должно выводиться сообщение о том, что массив (строка) пустые.

7) При реализации функций необходимо продемонстрировать использование параметров разных типов и различные способы организации функций (параметры по умолчанию, перегрузку функций, и .т.д.).

8) При обработке строки знаки препинания не должны удаляться, за исключением тех случаев, когда это требуется по условию задачи, например, при удалении предложения.

9) Рекомендуется при отладке программы сначала полностью отладить выполнение одной задачи и только после этого переходить к следующей.

10) Можно использовать класс Array и методы класса Array, класс String и методы класса String, StringBuilder и методы класса StringBuilder, класс Regex и методы класса Regex.

11) **Методы расширения коллекций в программе не используются.**

6. Требования к программе

1. Реализация основных функций задачи (создание, обработка в соответствии с вариантом, вывод полученных результатов).

2. Дополнительные функции (проверка правильности вводимых данных и т.д.)
3. Стилевое оформление программы.
4. Удобный интерфейс в виде двухуровневого текстового меню с выделением задач и подзадач.
5. Использование разных типов функций (перегрузка, параметры по умолчанию, функции с переменным числом параметров, рекурсивные функции и т.п.).
6. Обработка стандартных исключений.
7. Использование возможностей языка программирования, изучаемых самостоятельно (дополнительные баллы).

7. Содержание отчета

1. Описание этапа анализа (классы входных и выходных данных для каждой задачи).
2. Описание этапа проектирования (описание функций и их интерфейсов).
3. Код программы на C#.
4. Тесты с проверкой полноты по критериям черного ящика.

8. Примерные вопросы для защиты лабораторной работы

1. Как объявляется и инициализируется двумерный массив в C#? Приведите примеры синтаксиса объявления и инициализации.
2. Чем отличаются рваные массивы от обычных двумерных массивов?
3. Объясните принцип работы рваного массива.
4. Покажите пример объявления и инициализации рваного массива.
5. Какие методы класса System.Array вы знаете? Расскажите о назначении некоторых из этих методов.
6. Напишите пример использования метода Copy() для копирования массива.
7. Объясните разницу между обычным строковым литералом и буквальным строковым литералом в C#.
8. Какой префикс используется для обозначения буквального строкового литерала?
9. Почему иногда предпочтительнее использовать буквальный строковый литерал?
10. Расскажите о свойствах и методах класса String. Что делает метод CompareTo()? Чем полезен метод Split()?
11. Опишите процесс форматирования строк в C#, какие параметры можно задать в строке форматирования?
12. Представьте пример использования строки форматирования для вывода числа в денежном формате.
13. Что такое функция?
14. Что такое возвращаемое значение функции?
15. В чем разница между функцией, которая возвращает значение, и функцией без возвращаемого значения (void)?
16. Показать пример простой функции, которая принимает параметры и возвращает результат.
17. Способы передачи параметров в функцию.
18. Передача по значению и передача по ссылке. В чем отличие?
19. Зачем нужны модификаторы ref и out?
20. Что такое перегрузка методов? Привести пример перегруженного метода.

9. Критерии оценки выполнения программы

1. Формирование и обработка двумерного массива (с использованием функций) – 1 балл.
2. Формирование и обработка рваного массива (с использованием функций) – 2 балла.

3. Формирование и обработка строки (с использованием функций класса String) – 2 балла.
4. Оформление программы с учетом стайл-гайда – 1 балл.
5. Исправление ошибок при вводе – 1 балл.
6. Использование разных типов функций (перегрузка, параметры по умолчанию, функции с переменным числом параметров, рекурсивные функции и т.п.). – 1 балл
7. Использование регулярных выражений – 1 балл.