Практическая работа №5
Функции и массивы

1. Цель работы:

1) Получение практических навыков при работе с одномерными, многомерными и рваными массивами.

2) Получение практических навыков при работе с функциями, передаче данных в функции различными способами, получение результатов из функций различными способами.

3) 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;
```

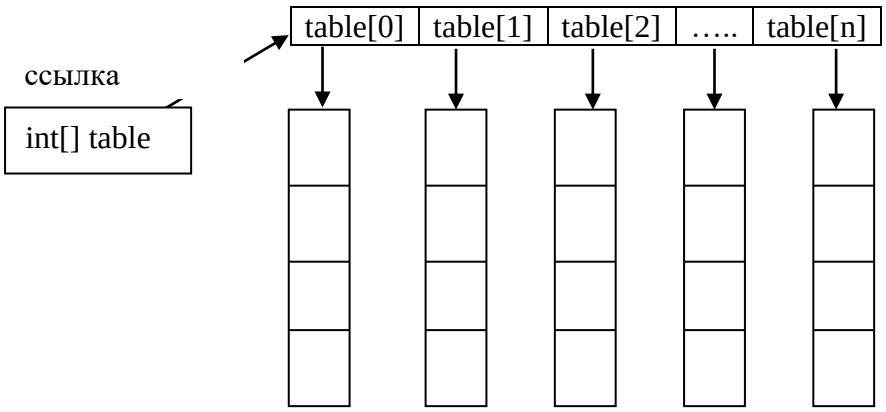

Рис. Выделение памяти под двумерный массив

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("Фрмирование 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);
}
```

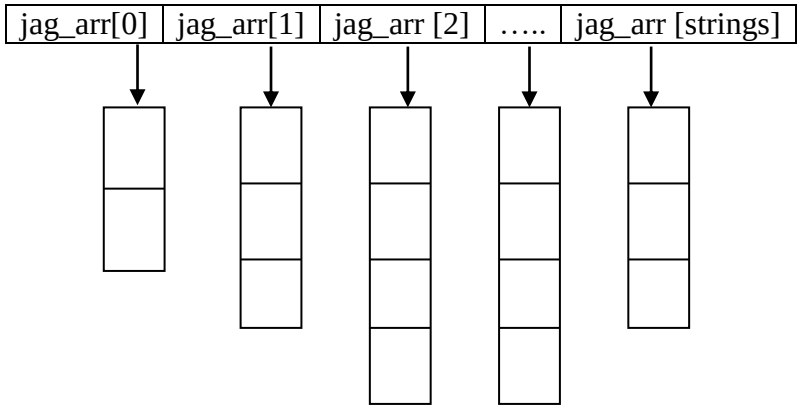

Рис. Выделение памяти под рваный массив

```
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. 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.

2.6. 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.

2.7. 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.

2.8. 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.

2.9. 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.

2.10. 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.

2.11. Использование меню для организации диалога с пользователем

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

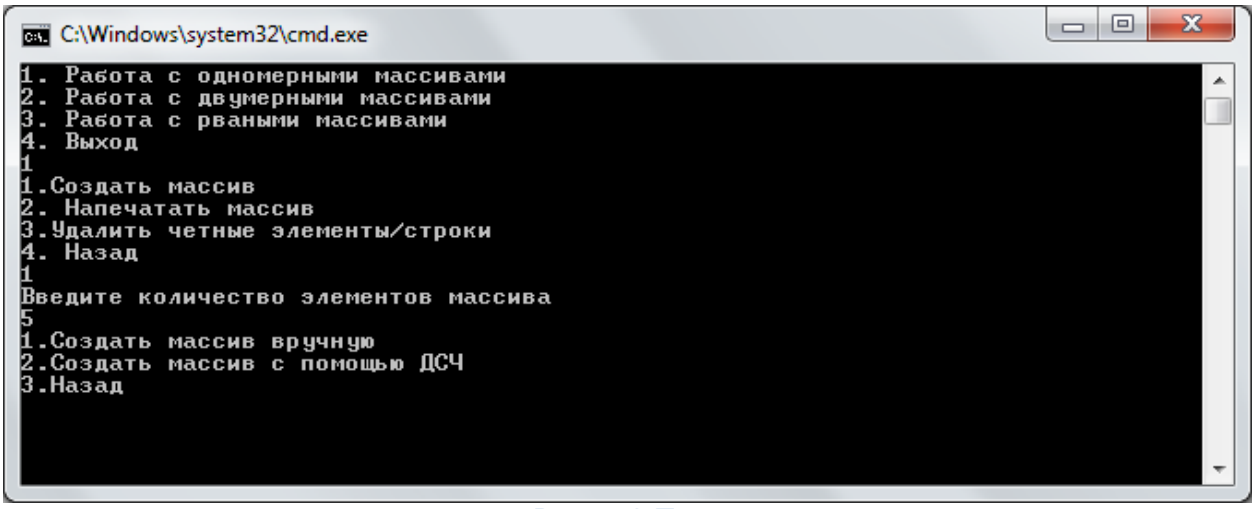

Рисунок 2. Пример меню

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 одномерный массив, заполнить его случайными числами и вывести на печать.
2. Выполнить указанное в варианте задание и вывести полученный массив на печать.
3. Сформировать динамический двумерный массив, заполнить его случайными числами и вывести на печать.
4. Выполнить указанное в варианте задание и вывести полученный массив на печать.
5. Сформировать динамический двумерный массив, заполнить его случайными числами и вывести на печать.

6. Выполнить указанное в варианте задание и вывести полученный массив на печать.

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

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 K 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 K2 включительно
7	Удалить все четные элементы	Добавить K строк в конец матрицы	Удалить первую строку, в которой встречаются нули
8	Удалить все элементы с четными индексами	Добавить K столбцов в конец матрицы	Удалить первую строку, в которой встречается заданное число K
9	Удалить все нечетные элементы	Добавить K строк в начало матрицы	Удалить строку с заданным номером
10	Удалить все элементы с нечетными индексами	Добавить K столбцов в начало матрицы	Удалить все строки с четными номерами
11	Добавить элемент в начало массива	Удалить строку с номером K	Добавить K строк, начиная с номера N
12	Добавить элемент в конец массива	Удалить столбец с номером K	Добавить K строк в конец массива
13	Добавить K элементов в начало массива	Удалить строки, начиная со строки K1 и до строки K2 включительно	Добавить строку с заданным номером
14	Добавить K элементов в конец массива	Удалить столбцы, начиная со столбца K1 и до столбца K2	Добавить строку в начало массива
15	Добавить K 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 К	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 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 возникновение исключительных ситуаций при вводе символов вместо цифр числа.

4) 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.

5) При попытке вывода пустого массива должно выводиться сообщение о том, что массив пустой.

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

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.
5. Использование разных типов функций (перегрузка, параметры по умолчанию, функции с переменным числом параметров, рекурсивные функции и т.п.).
6. Использование исключений.
7. Использование возможностей языка программирования, изучаемых самостоятельно.

7. Содержание отчета

1. Описание этапа анализа.
2. Описание этапа проектирования (описание функций и их интерфейсов).
3. Листинг программы.
4. Тесты с проверкой полноты по критериям черного и белого ящика.