Оглавление

Введение

Многомерные массивы применяются в тех случаях, когда обрабатываемые данные представляются в виде таблиц или совокупности таблиц. В первом случае используют двумерные массивы (другое название двумерного массива – матрица), во втором – трехмерные. Существуют массивы и большей размерности. Visual Basic 2005 позволяет создавать массивы, содержащие до 32 размерностей. Рассмотрим основные приемы обработки матриц.

1. Объявление двумерного массива

Описание двумерного массива практически не отличается от описания одномерного массива. Сначала указывается имя матрицы, затем – круглые скобки, между которыми ставится одна запятая. Она показывает Visual Basic 2005, что мы организуем массив, который будет иметь два индекса. Последним задается тип данных, к которому будут принадлежать все элементы матрицы. Существует два способа объявления матрицы.

Первый способ

```
Dim a(,) As Integer
```

Эта конструкция описывает двумерный целочисленный массив `a(,)` типа `Integer`, размеры которого заранее неизвестны. Изначально в нем нет ни одного элемента. Размеры этой матрицы будут определены позднее с помощью оператора `ReDim`.

Второй способ

```
Dim b(10, 18) As Single
```

Такая запись определяет двумерный массив `b(,)` типа `Single`, состоящий из 11 строк и 19 столбцов. Строки матрицы будут пронумерованы от 0 до 10, а столбцы – от 0 до 18. Обратите внимание, что при описании матрицы всегда сначала указывается номер последней строки, а потом – номер последнего столбца. Обратите внимание, что нумерация строк и столбцов матрицы всегда начинается с нуля независимо от способа ее объявления. Размеры этого массива тоже можно будет изменять в процессе выполнения программы с помощью оператора `ReDim`.

Объем памяти, занимаемой матрицей, вычисляется как произведение числа строк, числа столбцов и объема памяти, который занимает одна переменная указанного типа данных. Вычислим объем памяти, необходимый для хранения матрицы `b(,)`. В

Оглавление

этой матрице 11 строк, 19 столбцов. Один элемент матрицы занимает 4 байта памяти, так как имеет тип `Single`. Перемножаем эти числа и получаем: 11 * 19 * 4 = 836 байт.

При обращении к отдельному элементу двумерного массива необходимо в круглых скобках указать оба индекса, разделив их запятой. Обратите внимание, что всегда сначала указывается номер строки, а затем – номер столбца. Нумерация всегда начинается с нуля.

```
b(3, 6) = 7.5
```

Для обработки двумерных массивов применяется специальная конструкция из двух циклов, один из которых находится в теле другого. Такая конструкция называется вложенные циклы. Она записывается следующим образом.

```
For i = 0 to 10
    For j = 0 to 18
        A(i, j) = i + j
    Next
Next
```

На каждом шаге внешнего цикла (`For i = 0 to 10`) полностью выполняется внутренний цикл (`For j = 0 to 18`). Внешний цикл выполняется 11 раз, внутренний – 19. Следовательно, действие, находящееся в теле внутреннего цикла выполнится 11 * 19 = 209 раз. При организации вложенных циклов важно помнить, что счетчики внешнего и внутреннего циклов обязательно должны быть различными.

2. Ввод прямоугольной матрицы

Ввести матрицу – значит, задать ее размеры и указать значения всех ее элементов. Как и при обработке одномерного массива, значения элементов матрицы можно задавать двумя способами: вводя значения с клавиатуры или пользуясь генератором случайных чисел. Рассмотрим особенности программной реализации первого способа.

Задание значений элементов массива с клавиатуры – это самый распространенный способ ввода массива. Он состоит из двух этапов. На первом шаге указывается количество элементов в массиве и соответствующим образом переопределяется размер массива. На втором шаге организуется цикл, на каждом шаге которого вводится значение одного элемента. Рассмотрим особенности программной реализации этого алгоритма.

Оглавление

Сначала описывается целочисленная матрица `a(,)`. Так как ее размеры заранее неизвестны, то при описании матрицы номера последней строки и столбца не указываются.

```
Dim a(,) As Integer
```

Для работы с матрицей нам необходимо знать ее размеры. Они будет храниться в переменных `m` и `n`. Число строк – в переменной `m`, число столбцов – в переменной `n`. Обе этих переменных будут иметь целочисленный тип данных.

```
Dim m, n As Integer
```

Поскольку для обработки матрицы нам потребуется два цикла `For`, то нам необходимо описать два счетчика `i` и `j`. Очевидно, что обе эти переменных всегда будут иметь целый тип.

```
Dim i, j As Integer
```

Задание матрицы начинается с определения ее размеров. Мы просим пользователя указать число строк в матрице. Так как это количество может быть только положительным, то при вводе этого значения необходима проверка, которую мы организуем с помощью цикла `Do Loop Until`.

```
Do
    m = Val(InputBox("Введите количество строк"))
Loop Until m > 0
```

Аналогичным образом вводится число столбцов матрицы.

```
Do
    n = Val(InputBox("Введите количество столбцов"))
Loop Until n > 0
```

В Visual Basic 2005 нумерация элементов любого массива (в том числе и двумерного) всегда начинается с нуля. Следовательно, номера последней строки и последнего столбца будут на единицу меньше соответствующего количества. Поэтому уменьшаем значения переменных `m` и `n` на единицу. Теперь в них хранятся не количества строк и столбцов, а номера последней строки и последнего столбца матрицы.

```
m -= 1
n -= 1
```

Задаем размер матрицы `a(,)`, указывая в операторе `ReDim` номера последней строки и последнего столбца. При этом порядок перечисления переменных имеет значение.

```
ReDim a(m, n)
```

Оглавление

Для ввода значений элементов матрицы нам потребуется организовать вложенные циклы. Строки матрицы последовательно пронумерованы от 0 до m, а столбцы от 0 до n. Следовательно, счетчики обоих циклов должны изменяться в этих же диапазонах. Внешний цикл будем использовать для последовательной обработки строк матрицы, а внутренний – для работы с элементами, находящимися в пределах одной строки. В таких случаях говорят, что внешний цикл идет по строкам матрицы, а внутренний – по столбцам. В результате на i-м шаге внешнего цикла мы будем работать с i-й строкой матрицы, а на j-м шаге внутреннего цикла обрабатывать элемент, стоящий в i-й строке и j-м столбце.

```
For i = 0 To m
    For j = 0 To n
```

С помощью функции InputBox вводим значение элемента a(i, j). Так как вводимое значение является числом, то используем преобразование Val.

```
            a(i, j) = Val(InputBox("Введите элемент (" + _
                          Str(i) + "," + Str(j) + ")"))
    Next
Next
```

3. Вывод прямоугольной матрицы в окно списка и в текстовое поле

Существует много различных способов вывода матрицы. Рассмотрим два из их: вывод в окно списка и в вывод в текстовое поле.

При выводе матрицы в окно списка используется тот же прием, что и при выводе одномерного массива в текстовое поле. Сначала в некоторой дополнительной переменной формируется выводимая строка, которая затем выводится в окно списка. Переменная, предназначенная для хранения этой строки, всегда будет иметь тип String.

```
Dim s As String
```

Очищаем окно списка.

```
lstMatrix.Items.Clear()
```

Выводим поясняющий заголовок.

```
lstMatrix.Items.Add("Матрица")
```

Организуем цикл по всем строкам матрицы.

```
For i = 0 To m
```

Оглавление

Перед началом формирования очередной выводимой строки очищаем строковую переменную, записывая в нее пустую строку. Пустая строка обозначается кавычками, между которыми нет ни одного символа.

```
s = ""
```

Организуем цикл по столбцам матрицы. Он позволит нам перебрать все элементы в пределах `i`-й строки.

```
For j = 0 To n
```

К строковой переменной дописываем очередной элемент матрицы, преобразуя его к текстовому виду с помощью функции `Str`, и константу `vbTab`, которая позволит организовать вывод в несколько колонок.

```
        s += Str(a(i, j)) + vbTab
    Next
```

После завершения внутреннего цикла в строковой переменной записаны все элементы `i`-й строки матрицы. Выводим значение этой переменной в окно списка.

```
        lstMatrix.Items.Add(s)
Next
```

Вывод матрицы в текстовое поле требует специальной настройки данного элемента управления на этапе проектирования интерфейса. Мы будем выводить матрицу в текстовое поле с именем `txtMatrix`. Сначала помещаем текстовое поле на форму и задаем ему имя. По умолчанию текстовое поле предназначено для вывода однострочной информации, а нам требуется вывести матрицу, состоящую из нескольких строк. Поэтому для текстового поля `txtMatrix` мы изменим значение свойства `Multiline` и зададим ему значение `True`. Так мы включаем для текстового поля режим многострочного вывода. После этого можно изменить вертикальный размер текстового поля, сделав его достаточным для вывода матрицы.

Так как информацию в текстовое поля нельзя выводить по частям, то для вывода матрицы нам потребуется дополнительная переменная типа `String`, в которой будет формироваться выводимый текст. Опишем эту переменную.

```
Dim s As String
```

В качестве начального значения запишем в переменную поясняющий заголовок и константу `vbNewLine`, которая обеспечивает переход на новую строку.

```
s = "Матрица" + vbNewLine
```

Организуем цикл по всем строкам матрицы.

Оглавление

```
For i = 0 To m
```

Организуем цикл по столбцам матрицы. Он позволит нам перебрать все элементы в пределах `i`-й строки.

```
        For j = 0 To n
```

К строковой переменной дописываем очередной элемент матрицы, преобразуя его к текстовому виду с помощью функции `Str`, и константу `vbTab`, которая позволит организовать вывод в несколько колонок.

```
            s += Str(a(i, j)) + vbTab
        Next
```

После завершения внутреннего цикла в строковой переменной записаны все элементы `i`-й строки матрицы. Дописываем к этой переменной константу `vbNewLine`, чтобы следующая строка матрицы выводилась на новой строке текстового поля.

```
        s += vbNewLine
Next
```

После завершения внешнего цикла в переменной `s` записаны все элементы матрицы. Нам остается только вывести эту переменную в текстовое поле.

```
txtMatrix.Text = s
```

Пример работы программы приведен на рис. 1.

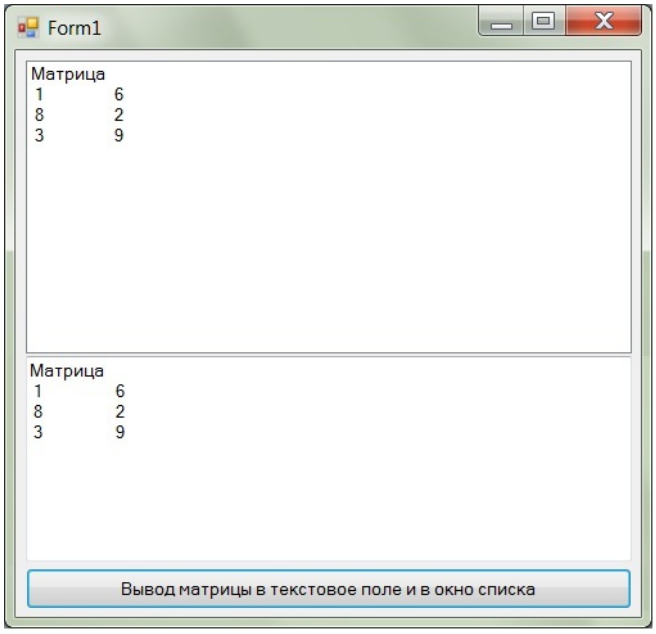

Рис. 1. Пример работы программы вывода прямоугольной матрицы в окно списка и текстовое поле

Оглавление

4. Поиск максимального элемента матрицы

Для поиска максимального элемента матрицы и определения его индексов нам потребуются три дополнительные переменные. В первой из них будем хранить значение максимального элемента, а во второй – номер строки, к которой он находится, а в третьей – номер столбца. Поиск максимального элемента в матрице традиционно начинают с элемента, стоящего в нулевой строке и нулевом столбце. Затем организуют вложенные циклы, которые позволят проанализировать все элементы матрицы. Если значение какой-либо элемента матрицы окажется больше ранее найденного максимума, то значение максимума необходимо обновить, сделав равным этому элементу. При этом сразу запоминаются оба его индекса. Рассмотрим особенности программной реализации этого алгоритма.

Объявляем необходимые переменные. Переменная `max` предназначена для хранения значения максимального элемента. Ее тип всегда должен совпадать с типом элементов матрицы. В переменной `imax` мы будем запоминать номер строки, в которой находится максимальный элемент матрицы, а в переменной `jmax` мы будем хранить номер столбца, в котором находится максимальный элемент. Эти две переменные всегда будут иметь целочисленный тип.

```
Dim max, imax, jmax As Integer
```

Поиск максимума мы начинаем с элемента, стоящего в нулевой строке и нулевом столбце матрицы.

```
max = a(0, 0)
```

В переменные `imax` и `jmax` мы записываем соответственно номера строки и столбца, которые равны нулю.

```
imax = 0
jmax = 0
```

Организуем цикл по всем строкам матрицы.

```
For i = 0 To m
```

Организуем цикл по столбцам матрицы. Он позволит нам перебрать все элементы в пределах `i`-й строки.

```
    For j = 0 To n
```

Анализируем очередной элемент матрицы.

```
        If a(i, j) > max Then
```

Оглавление

Если его значение больше, чем значение ранее найденного максимума, то необходимо обновить значение максимума (переменной `max`), записав в него значение текущего элемента.

```
                max = a(i, j)
```

Сразу же запоминаем индексы нового максимума. Номер строки, в которой находится данный элемент матрицы, хранится в переменной `i`, а номер столбца – в переменной `j`.

```
                imax = i
                jmax = j
            End If
        Next
    Next
```

После завершения вложенных циклов нам остается только вывести полученные результаты в окно списка. Сначала выведем горизонтальную черту, чтобы зрительно отделить исходные данные от полученных результатов.

```
    lstMatrix.Items.Add("-----------------------------")
```

Заем выведем максимальное значение и его индексы.

```
    lstMatrix.Items.Add("Максимум:" + Str(max))
    lstMatrix.Items.Add("Строка:" + Str(imax))
    lstMatrix.Items.Add("Столбец:" + Str(jmax))
```

Полный текст программы представлен в приложении 1. Пример работы программы приведен на рис. 2.

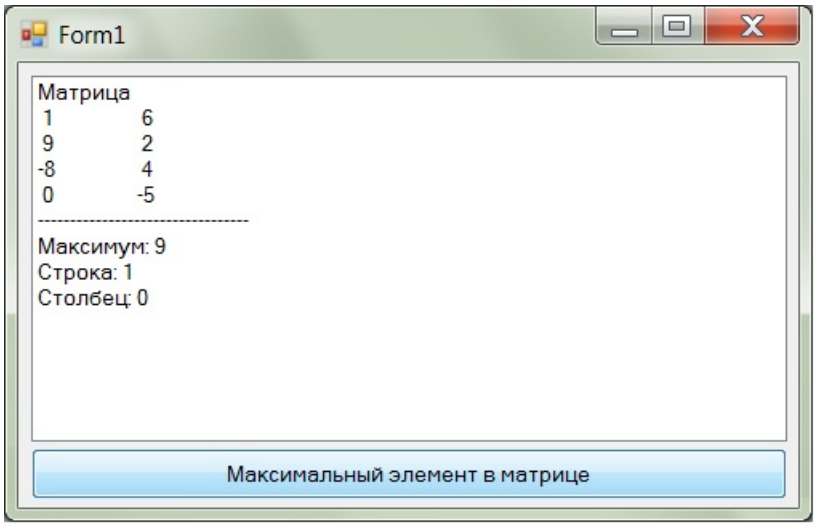

Рис. 2. Пример работы программы поиска максимального элемента в прямоугольной матрице

Оглавление

5. Обработка матрицы по строкам

Обработка прямоугольной матрицы может вестись двумя способами: по строкам и по столбцам. Как правило, способ обработки определяется условием решаемой задачи. При обработке по строкам вложенные циклы организуются таким образом, чтобы за один шаг внешнего цикла полностью обрабатывалась одна строка матрицы. В этом случае внешний цикл будет идти по строкам матрицы, а внутренний – по столбцам. В качестве примера рассмотрим задачу вычисления среднего арифметического четных элементов в каждой строке матрицы.

Для решения этой задачи необходимо в каждой строке матрицы вычислить свою сумму и количество четных элементов. Затем, если это возможно, найти среднее арифметическое. Если же в какой-то строке матрицы нет четных элементов, то вместо среднего арифметического мы будем печатать поясняющее сообщение. Результаты работы программы будем выводить в окно списка.

Начнем с объявления необходимых переменных. Переменные `sum` и `kol`, имеющие целочисленный тип данных, предназначены для хранения суммы и количества четных элементов в одной строке матрицы.

```
Dim sum, kol As Integer
```

Так как среднее арифметическое получается в результате деления, то соответствующая переменная всегда будет иметь рациональный тип данных.

```
Dim sred As Single
```

Для каждой строки матрицы мы будем получать отдельный результат. Поэтому удобно совместить процессы вычисления и вывода в одном цикле. Чтобы результаты вычислений зрительно отделялись от исходных данных, выведем горизонтальную черту.

```
lstMatrix.Items.Add("----------------------------")
```

Результаты вычислений мы будем выводить в две колонки. В первой будет указываться номер обрабатываемой строки матрицы, а во второй – найденное значение среднего арифметического четных элементов. Выводим поясняющий заголовок. Константа `vbTab` позволяет организовать вывод в две колонки.

```
lstMatrix.Items.Add("Строка" + vbTab + _
                    "Сред. арифм.")
```

Оглавление

Так как по условию задачи требуется, чтобы каждая строка матрицы обрабатывалась отдельно, то обработку матрицы будем вести по строкам. Для этого организуем внешний цикл по строкам, а внутренний – по столбцам.

```
For i = 0 To m
```

За один шаг внешнего цикла мы будем полностью обрабатывать одну строку матрицы. Для этого мы будем использовать внутренний цикл. Следовательно, начальные значения для суммы и количества необходимо задать до него.

```
sum = 0
kol = 0
```

Организуем внутренний цикл. Нам необходимо обработать все элементы, находящиеся в `i`-й строке матрицы. Для этого мы должны пройтись по всем столбцам матрицы. Соответственно, внутренний цикл будет по столбцам.

```
For j = 0 To n
```

На каждом шаге цикла анализируем элемент матрицы.

```
    If a(i, j) Mod 2 = 0 Then
```

Если его значение при делении на 2 дает в остатке ноль, значит, оно четное, и данный элемент надо добавить к сумме. При этом количество четных элементов в данной строке матрицы увеличивается на единицу.

```
        sum += a(i, j)
        kol += 1
    End If
Next
```

После завершения внутреннего цикла можно приступать к вычислению среднего арифметического четных элементов в одной строке матрицы. Сначала анализируем количество четных элементов.

```
If kol = 0 Then
```

Если количество четных элементов равно нулю, значит, их в рассматриваемой строке нет ни одного четного элемента. Следовательно, вычислить их среднее арифметическое невозможно. Поэтому вместо ответа выводим номер строки и поясняющее сообщение.

```
    lstMatrix.Items.Add(Str(i) + vbTab + _
                                "В строке нет четных")
Else
```

Иначе мы делим сумму четных элементов на их количество и получаем искомое значение.

Оглавление

```
                sred = sum / kol
```

В окно списка выводим номер строки и найденное значение среднего арифметического четных элементов в этой строке.

```
                lstMatrix.Items.Add(Str(i) + vbTab + _
                                    Str(sred))
            End If
        Next
```

Полный текст программы представлен в приложении 2. Пример работы программы приведен на рис. 3.

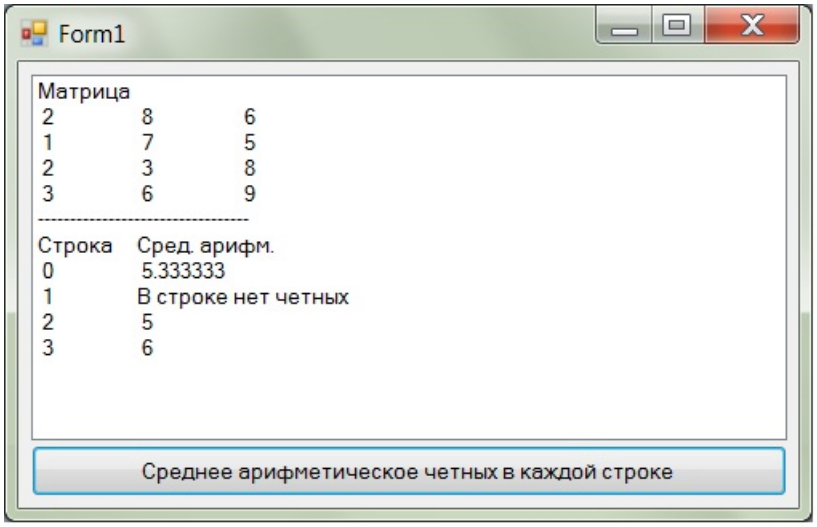

Рис. 3. Пример работы программы поиска среднего арифметического четных элементов в каждой строке прямоугольной матрицы

6. Обработка матрицы по столбцам

Обработка матрицы по столбцам практически ничем не отличается от обработки матрицы по строкам. Единственное различие – в порядке вложенных циклов. При обработке матрицы по столбцам внешний цикл организуют по столбцам, а внутренний – по строкам. В качестве примера рассмотрим задачу поиска минимального элемента в каждом столбце матрицы. Полученные результаты запишем в одномерный массив, который затем выведем в окно списка.

Решение задачи начнем с описания переменных. Нам потребуется переменная для хранения минимального значения в отдельном столбце матрицы. Очевидно, что она будет иметь такой же тип данных, что и элементы исходной матрицы.

```
        Dim min As Integer
```

Оглавление

Так как все найденные значения будут записываться в одномерный массив, то нам необходимо его объявить.

```
Dim b() As Integer
```

Сразу зададим размер результирующего массива. Поскольку мы ищем минимум в каждом столбце по отдельности, то количество результатов всегда будет совпадать с количеством столбцов матрицы. Таким образом, номер последнего элемента в массиве результатов будет равен номеру последнего столбца матрицы.

```
ReDim b(n)
```

Организуем вложенные циклы. Так как по условию задачи требуется обработать каждый столбец по отдельности, то внешний цикл будет идти по столбцам матрицы.

```
For j = 0 To n
```

Каждый столбец матрицы мы будем рассматривать как одномерный массив. Поэтому задача поиска минимального элемента в одном столбце матрицы решается точно также как задача поиска минимального элемента в одномерном массиве. Поиск начинается с нулевого элемента столбца. Этот элемент находится в нулевой строке и `j`-м столбце, где `j` – счетчик внешнего цикла.

```
min = a(0, j)
```

Обрабатываем все остальные элементы j-го столбца. Для этого мы должны пройти по всем строкам, входящим в этот столбец. Следовательно, внутренний цикл организуем по стокам. Чтобы не сравнивать нулевой элемент столбца с самим собой внутренний цикл начнем с первой строки.

```
For i = 1 To m
```

Анализируем очередной элемент столбца.

```
If a(i, j) < min Then
```

Если его значение меньше ранее найденного минимума, значит, значение минимума надо обновить, записав в соответствующую переменную значение текущего элемента матрицы.

```
        min = a(i, j)
    End If
Next
```

После завершения внутреннего цикла в переменной `min` хранится значение минимального элемента в `j`-м столбце. Это значение мы должны записать в результирующий массив на `j`-ю позицию.

```
b(j) = min
```

Оглавление

```
Next
```

После завершения вложенных циклов результирующий массив полностью сформирован, и мы должны вывести его в окно списка. Чтобы зрительно отделить исходные данные от полученных результатов выведем горизонтальную черту.

```
lstMatrix.Items.Add("-----------------------------")
```

Выводим поясняющий заголовок. В первой колонке мы будем печатать номер столбца, а во втором – значение минимального элемента в этом столбце. Константа `vbTab` позволит нам организовать вывод в две колонки.

```
lstMatrix.Items.Add("Столбец" + vbTab + "Минимум")
```

Организуем цикл для вывода массива `b()`. Так как количество элементов в этом массиве совпадает с количеством столбцов в матрице, то значение счетчика будет изменяться в диапазоне от 0 до `n`, где `n` – номер последнего столбца матрицы.

```
For i = 0 To n
```

На каждом шаге цикла выводим номер элемента массива (он же – номер столбца матрицы) и значение элемента массива – это минимальный элемент в данном столбце матрицы.

```
    lstMatrix.Items.Add(Str(i) + vbTab + Str(b(i)))
Next
```

Полный текст программы представлен в приложении 3. Пример работы программы приведен на рис. 4.

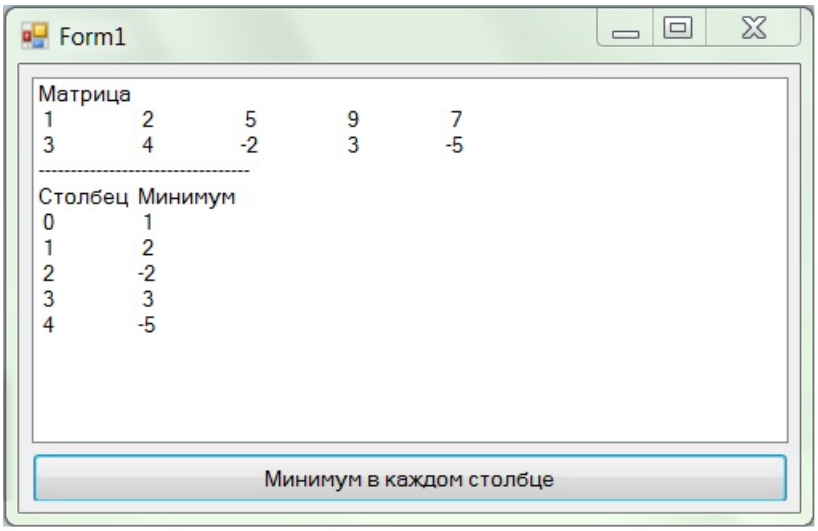

Рис. 4. Пример работы программы поиска минимального элемента в каждом столбце прямоугольной матрицы

Оглавление

7. Обработка квадратных матриц

Квадратной называют матрицу, у которой число строк равно числу столбцов. В общем случае обработка квадратных матриц принципиально не отличается от обработки прямоугольной матрицы. Для нее также используют вложенные циклы, с той лишь разницей, что диапазоны значений для внешнего и внутреннего счетчиков будут одинаковыми. Посмотрим, как изменяется текст программы ввода матрицы и вывода ее в окно списка при переходе от прямоугольной матрицы к квадратной.

Так как размеры у квадратной матрицы совпадают, то для их хранения достаточно одной переменной. В квадратной матрице `n` строк и `n` столбцов. Следовательно, вместо двух блоков ввода (ввод числа строк и ввод числа столбцов) будет один – ввод размера матрицы.

```
Do
    n = Val(InputBox("Введите размер матрицы"))
Loop Until n > 0
```

Нумерация элементов в квадратной матрице идет также как и в прямоугольной – с нуля. Поэтому номера последней строки и столбца будут на единицу меньше соответствующего размера.

```
n -= 1
```

Выделяем память для хранения квадратной матрицы.

```
ReDim a(n, n)
```

Организуем вложенные циклы для ввода элементов матрицы. Диапазоны значений счетчиков у этих циклов совпадают, так как совпадают размеры матрицы.

```
For i = 0 To n
    For j = 0 To n
```

Вводим очередной элемент матрицы.

```
        a(i, j) = Val(InputBox("Введите элемент (" + _
                        Str(i) + "," + Str(j) + ")"))
    Next
Next
```

С завершением вложенных циклов завершается и процесс ввода значений элементов квадратной матрицы. Начинаем процесс вывода матрицы в окно списка. Сначала это окно необходимо очистить от предыдущих результатов работы программы.

```
lstMatrix.Items.Clear()
```

Затем мы выводим поясняющий заголовок.

Оглавление

```
lstMatrix.Items.Add("Матрица")
```

Организуем вложенные циклы для вывода матрицы. Диапазоны значений счетчиков у этих циклов совпадают, так как совпадают размеры матрицы. В остальном процесс вывода квадратной матрицы ничем не отличается от вывода прямоугольной матрицы.

```
For i = 0 To n
    s = ""
    For j = 0 To n
        s += Str(a(i, j)) + vbTab
    Next
    lstMatrix.Items.Add(s)
Next
```

При обработке квадратных матриц можно использовать все алгоритмы, разработанные для прямоугольных матриц. Но существует ряд задач, характерных именно для квадратных матриц. Это задачи обработки элементов, находящихся на диагоналях матрицы или в одном из ее треугольников.

В квадратной матрице выделяют главную и побочную диагонали. Главная диагональ (см. рис. 5) идет из верхнего левого угла матрицы в правый нижний угол, то есть от элемента с индексами (0, 0) до элемента с индексами (`n`, `n`), где `n` – номер последнего столбца и строки матрицы. На каждой строке матрицы находится ровно один элемент, стоящий на главной диагонали. Для этого элемента всегда будет выполняться условие `i = j`, где `i` – номер строки, а `j` – номер столбца.

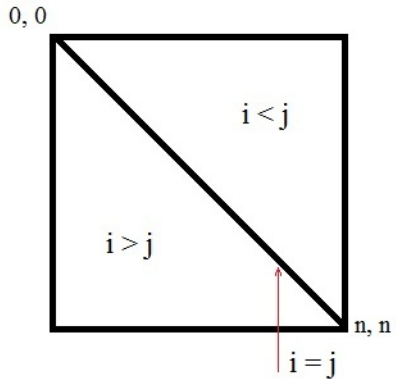

Рис. 5. Главная диагональ квадратной матрицы

Главная диагональ разбивает матрицу на два треугольника. Все элементы верхнего треугольника расположены над главной диагональю. Для них выполняется

Оглавление

условие `i < j`, где `i` – номер строки, а `j` – номер столбца. Все элементы нижнего треугольника расположены под главной диагональю. Для них выполняется условие `i > j`, где `i` – номер строки, а `j` – номер столбца. Как правило, элементы, стоящие на диагонали, не включаются ни в один треугольник. Если же по условию задачи требуется включить диагональные элементы в один из треугольников, то в соответствующем условии строгое неравенство заменяется на нестрогое. Условия главной диагонали и соответствующих треугольников проиллюстрированы на рис. 5.

Побочная диагональ (см. рис. 6) идет из верхнего правого угла матрицы в левый нижний угол, то есть от элемента с индексами (0, `n`) до элемента с индексами (`n`, 0), где `n` – номер последнего столбца и строки матрицы. На каждой строке матрицы находится ровно один элемент, стоящий на побочной диагонали. Для этого элемента всегда будет выполняться условие `i + j = n`, где `i` – номер строки, `j` – номер столбца, `n` – номер последней строки и последнего столбца матрицы.

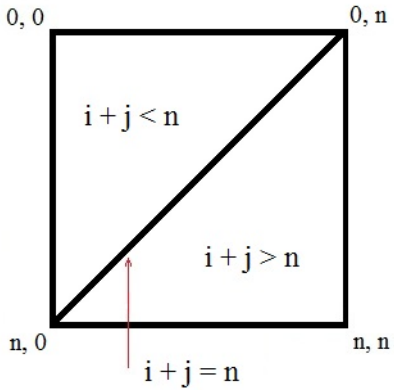

Рис. 6. Побочная диагональ квадратной матрицы

Побочная диагональ разбивает матрицу на два треугольника. Все элементы верхнего треугольника расположены над побочной диагональю. Для них выполняется условие `i + j < n`, где `i` – номер строки, `j` – номер столбца, `n` – номер последней строки и последнего столбца матрицы. Все элементы нижнего треугольника расположены под побочной диагональю. Для них выполняется условие `i + j > n`, где `i` – номер строки, `j` – номер столбца, `n` – номер последней строки и последнего столбца матрицы. Как правило, элементы, стоящие на диагонали, не включаются ни в один треугольник. Если же по условию задачи требуется включить диагональные элементы в один из треугольников, то в соответствующем условии строгое неравенство

Оглавление

заменяется на нестрогое. Условия побочной диагонали и соответствующих треугольников проиллюстрированы на рис. 6.

Рассмотрим особенности алгоритма обработки диагональных элементов на примере решения задачи определения суммы элементов, стоящих на главной диагонали, и произведения элементов, стоящих на побочной диагонали. Полученные результаты будем выводить в окно списка.

Объявляем переменные для хранения суммы и произведения.

```
Dim sum, proiz As Integer
```

И задаем им начальные значения.

```
sum = 0
proiz = 1
```

Так как в каждой строке находится ровно один элемент с каждой диагонали, то для их обработки нам достаточно одного цикла, поскольку, зная номер строки, номер столбца можно вычислить, пользуясь условием соответствующей диагонали. Организуем цикл по всем строкам матрицы.

```
For i = 0 To n
```

Для элементов главной диагонали выполняется условие `i = j`, то есть номер строки совпадает с номером столбца. Следовательно, если элемент стоит на главной диагонали и находится в строке с номером `i`, то он будет стоять в столбце с номером `i`. Таким образом, любой элемент с главной диагонали матрицы `a(,)` можно обозначить `a(i, i)`. Добавляем этот элемент к итоговой сумме.

```
    sum += a(i, i)
```

Для элементов побочной диагонали выполняется условие `i + j = n`. Номер строки `i` нам известен, равно как и значение переменной `n`. Следовательно, мы можем из этого условия выразить значение переменной `j`.

```
j = n - i
```

Тогда, если элемент стоит на побочной диагонали и находится в строке с номером `i`, то он будет стоять в столбце с номером `n - i`. Таким образом, любой элемент с побочной диагонали матрицы `a(,)` можно обозначить `a(i, n - i)`. Включаем этот элемент в искомое произведение.

```
    proiz *= a(i, n - i)
Next
```

После завершения цикла печатаем полученные результаты. Сначала выводим горизонтальную черту, чтобы зрительно отделить исходные данные от результатов.

Оглавление

```
lstMatrix.Items.Add("-----------------------------")
```

Затем печатаем значения суммы и произведения с поясняющим текстом.

```
lstMatrix.Items.Add("Сумма = " + Str(sum))
lstMatrix.Items.Add("Произведение = " + Str(proiz))
```

Полный текст программы представлен в приложении 4. Пример работы программы приведен на рис. 7.

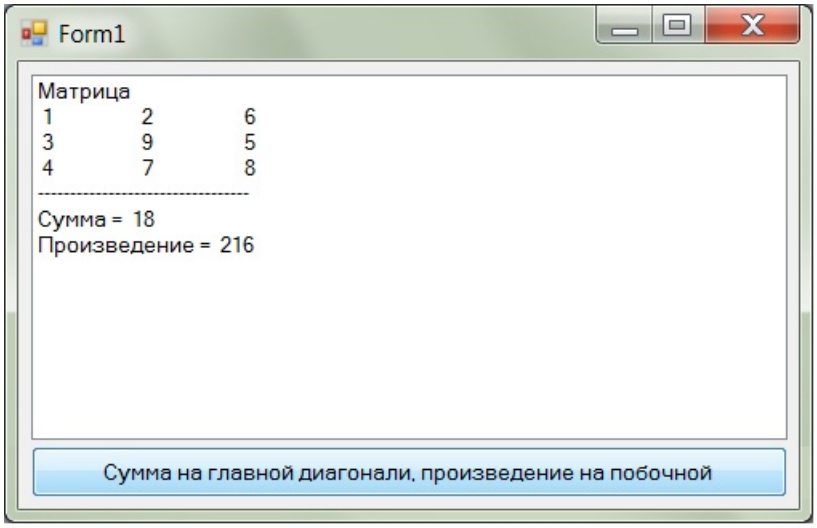

Рис. 7. Пример работы программы вычисления суммы элементов, стоящих на главной диагонали квадратной матрицы, и произведения элементов, стоящих на побочной диагонали квадратной матрицы

Теперь рассмотрим особенности алгоритма обработки элементов, принадлежащих к какому-нибудь треугольнику. В качестве примера возьмем задачу поиска минимального из элементов, стоящих выше главной диагонали, и максимального из элементов, стоящих ниже побочной диагонали. Помимо самих значений определим еще индексы искомых элементов. Полученные результаты будем выводить в окно списка.

Описываем переменные для хранения минимального и максимального значений, а также их индексов.

```
Dim min, imin, jmin As Integer
Dim max, imax, jmax As Integer
```

Зададим начальное значение для минимума. Им может быть любой элемент, который всегда будет принадлежать к нужному треугольнику. Удобнее всего брать элемент, который расположен в вершине прямого угла (см. рис. 5 и рис. 6). Для треугольника,

Оглавление

расположенного выше главной диагонали, это будет элемент, находящийся в правом верхнем углу – `a(0, n)`.

```
min = a(0, n)
```

Задаем соответствующие начальные значения индексов.

```
imin = 0
jmin = n
```

Рассуждая аналогичным образом, зададим начальное значение для максимума. Так как нужный треугольник расположен ниже побочной диагонали, то это будет элемент, находящийся в правом нижнем углу – `a(n, n)`.

```
max = a(n, n)
```

Задаем соответствующие начальные значения индексов.

```
imax = n
jmax = n
```

Организуем вложенные циклы для обработки всех элементов матрицы. Внешний цикл будет идти по строкам, а внутренний – по столбцам.

```
For i = 0 To n
    For j = 0 To n
```

На каждом шаге цикла проверяем, принадлежит ли очередной элемент матрицы к треугольнику, расположенному выше главной диагонали, и меньше ли значение этого элемента, чем ранее найденный минимум.

```
            If i < j And a(i, j) < min Then
```

Если на оба вопроса дан положительный ответ, значит надо обновить значение минимума, записав в соответствующую переменную значение анализируемого элемента матрицы.

```
                min = a(i, j)
```

При этом необходимо еще запомнить индексы текущего элемента.

```
                imin = i
                jmin = j
            End If
```

Теперь проверяем, принадлежит ли очередной элемент матрицы к треугольнику, расположенному ниже побочной диагонали, и больше ли значение этого элемента, чем ранее найденный максимум.

```
            If i + j > n And a(i, j) > max Then
```

Оглавление

Если на оба вопроса дан положительный ответ, значит надо обновить значение максимума, записав в соответствующую переменную значение анализируемого элемента матрицы.

```
                    max = a(i, j)
```

При этом необходимо еще запомнить индексы текущего элемента.

```
                    imax = i
                    jmax = j
                End If
            Next
        Next
```

После завершения вложенных циклов в соответствующих переменных хранится искомая информация. Нам остается только вывести ее в окно списка. Сначала выводим горизонтальную строку, чтобы зрительно отделить исходные данные от полученных результатов.

```
        lstMatrix.Items.Add("-----------------------------")
```

Затем печатаем поясняющий текст и значения необходимых переменных.

```
        lstMatrix.Items.Add("Минимум над глав. диагональю")
        lstMatrix.Items.Add("Значение:" + Str(min))
        lstMatrix.Items.Add("Строка:" + Str(imin))
        lstMatrix.Items.Add("Столбец:" + Str(jmin))
        lstMatrix.Items.Add("Максимум под побоч. диагональю")
        lstMatrix.Items.Add("Значение:" + Str(max))
        lstMatrix.Items.Add("Строка:" + Str(imax))
        lstMatrix.Items.Add("Столбец:" + Str(jmax))
```

Полный текст программы представлен в приложении 5. Пример работы программы приведен на рис. 8.

Оглавление

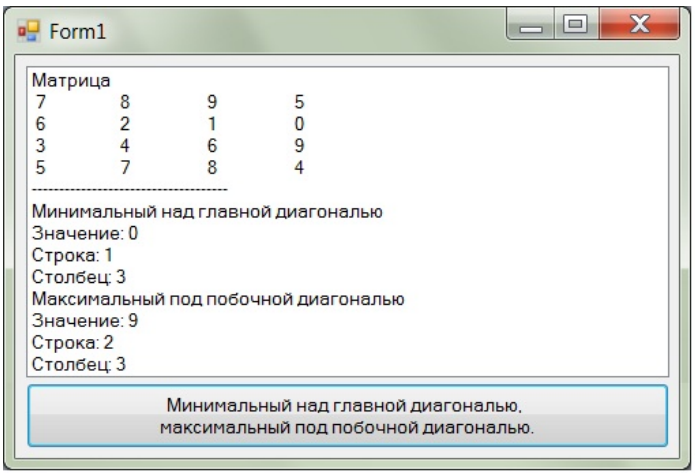

Рис. 8. Пример работы программы поиска минимального из элементов, стоящих выше главной диагонали квадратной матрицы, и максимального из элементов, стоящих ниже побочной диагонали квадратной матрицы

Сформулируем основные правила обработки элементов матрицы, стоящих на любой диагонали или принадлежащих к одному из треугольников.

Для обработки элементов, стоящих на любой диагонали, достаточно одного цикла.

Для обработки элементов, принадлежащих к одному из треугольников, необходимо использовать вложенные циклы.

Приложение 1

Дана прямоугольная целочисленная матрица. Размеры матрицы и значения ее элементов вводятся с клавиатуры. Найти максимальный элемент в матрице и его индексы. Исходную матрицу и полученные результаты вывести в окно списка.

```
Dim a(,) As Integer
Dim m, n As Integer
Dim i, j As Integer
Dim max, imax, jmax As Integer
Dim s As String
Do
    m = Val(InputBox("Введите количество строк"))
Loop Until m > 0
Do
    n = Val(InputBox("Введите количество столбцов"))
```

Оглавление

```
Loop Until n > 0
m -= 1
n -= 1
ReDim a(m, n)
For i = 0 To m
    For j = 0 To n
        a(i, j) = Val(InputBox("Введите элемент (" + _
                    Str(i) + "," + Str(j) + ")"))
    Next
Next
lstMatrix.Items.Clear()
lstMatrix.Items.Add("Матрица")
For i = 0 To m
    s = ""
    For j = 0 To n
        s += Str(a(i, j)) + vbTab
    Next
    lstMatrix.Items.Add(s)
Next
max = a(0, 0)
imax = 0
jmax = 0
For i = 0 To m
    For j = 0 To n
        If a(i, j) > max Then
            max = a(i, j)
            imax = i
            jmax = j
        End If
    Next
Next
lstMatrix.Items.Add("--------------------------------")
lstMatrix.Items.Add("Максимум:" + Str(max))
lstMatrix.Items.Add("Строка:" + Str(imax))
lstMatrix.Items.Add("Столбец:" + Str(jmax))
```

Оглавление

Приложение 2

Дана прямоугольная целочисленная матрица. Размеры матрицы и значения ее элементов вводятся с клавиатуры. В каждой строке матрицы найти среднее арифметическое четных чисел. Если в какой-нибудь строке нельзя вычислить среднее арифметическое, то вывести поясняющее сообщение. Исходную матрицу и полученные результаты вывести в окно списка.

```
Dim a(,) As Integer
Dim m, n As Integer
Dim i, j As Integer
Dim s As String
Dim sum, kol As Integer
Dim sred As Single
Do
    m = Val(InputBox("Введите количество строк"))
Loop Until m > 0
Do
    n = Val(InputBox("Введите количество столбцов"))
Loop Until n > 0
m -= 1
n -= 1
ReDim a(m, n)
For i = 0 To m
    For j = 0 To n
        a(i, j) = Val(InputBox("Введите элемент (" + _
                    Str(i) + "," + Str(j) + ")"))
    Next
Next
lstMatrix.Items.Clear()
lstMatrix.Items.Add("Матрица")
For i = 0 To m
    s = ""
    For j = 0 To n
        s += Str(a(i, j)) + vbTab
    Next
```

Оглавление

```
    lstMatrix.Items.Add(s)
Next
lstMatrix.Items.Add("--------------------------------")
lstMatrix.Items.Add("Строка" + vbTab + "Сред. арифм.")
For i = 0 To m
    sum = 0
    kol = 0
    For j = 0 To n
        If a(i, j) Mod 2 = 0 Then
            sum += a(i, j)
            kol += 1
        End If
    Next
    If kol = 0 Then
        lstMatrix.Items.Add(Str(i) + vbTab + _
                            "В строке нет четных")
    Else
        sred = sum / kol
        lstMatrix.Items.Add(Str(i) + vbTab + Str(sred))
    End If
Next
```

Приложение 3

Дана прямоугольная целочисленная матрица. Размеры матрицы и значения ее элементов вводятся с клавиатуры. Найти минимальный элемент в каждом столбце матрицы. Полученные результаты записать в одномерный массив. Исходную матрицу и результирующий массив вывести в окно списка.

```
Dim a(,) As Integer
Dim m, n As Integer
Dim i, j As Integer
Dim s As String
Dim min As Integer
Dim b() As Integer
Do
    m = Val(InputBox("Введите количество строк"))
```

Оглавление

```
Loop Until m > 0
Do
    n = Val(InputBox("Введите количество столбцов"))
Loop Until n > 0
m -= 1
n -= 1
ReDim a(m, n)
For i = 0 To m
    For j = 0 To n
        a(i, j) = Val(InputBox("Введите элемент (" + _
                    Str(i) + "," + Str(j) + ")"))
    Next
Next
lstMatrix.Items.Clear()
lstMatrix.Items.Add("Матрица")
For i = 0 To m
    s = ""
    For j = 0 To n
        s += Str(a(i, j)) + vbTab
    Next
    lstMatrix.Items.Add(s)
Next
ReDim b(n)
For j = 0 To n
    min = a(0, j)
    For i = 1 To m
        If a(i, j) < min Then
            min = a(i, j)
        End If
    Next
    b(j) = min
 Next
 lstMatrix.Items.Add("--------------------------------")
 lstMatrix.Items.Add("Столбец" + vbTab + "Минимум")
 For i = 0 To n
```

Оглавление

```
    lstMatrix.Items.Add(Str(i) + vbTab + Str(b(i)))
Next
```

Приложение 4

Дана квадратная целочисленная матрица. Размер матрицы и значения ее элементов вводятся с клавиатуры. Найти сумму элементов, стоящих на главной диагонали, и произведение элементов, стоящих на побочной диагонали. Исходную матрицу и полученные результаты вывести в окно списка.

```
Dim a(,) As Integer
Dim n As Integer
Dim i, j As Integer
Dim s As String
Dim sum, proiz As Integer
Do
    n = Val(InputBox("Введите размер матрицы"))
Loop Until n > 0
n -= 1
ReDim a(n, n)
For i = 0 To n
    For j = 0 To n
        a(i, j) = Val(InputBox("Введите элемент (" + _
                    Str(i) + "," + Str(j) + ")"))
    Next
Next
lstMatrix.Items.Clear()
lstMatrix.Items.Add("Матрица")
For i = 0 To n
    s = ""
    For j = 0 To n
        s += Str(a(i, j)) + vbTab
    Next
    lstMatrix.Items.Add(s)
Next
sum = 0
proiz = 1
```

Оглавление

```
For i = 0 To n
    sum += a(i, i)
    proiz *= a(i, n - i)
Next
lstMatrix.Items.Add("--------------------------------")
lstMatrix.Items.Add("Сумма = " + Str(sum))
lstMatrix.Items.Add("Произведение = " + Str(proiz))
```

Приложение 5

Дана квадратная целочисленная матрица. Размер матрицы и значения ее элементов вводятся с клавиатуры. Найти минимальный из элементов, стоящих выше главной диагонали, и его индексы. Найти максимальный из элементов, стоящих ниже побочной диагонали, и его индексы. Исходную матрицу и полученные результаты вывести в окно списка.

```
Dim a(,) As Integer
Dim n As Integer
Dim i, j As Integer
Dim s As String
Dim min, imin, jmin As Integer
Dim max, imax, jmax As Integer
Do
    n = Val(InputBox("Введите размер матрицы"))
Loop Until n > 0
n -= 1
ReDim a(n, n)
For i = 0 To n
    For j = 0 To n
        a(i, j) = Val(InputBox("Введите элемент (" + _
                    Str(i) + "," + Str(j) + ")"))
    Next
Next
lstMatrix.Items.Clear()
lstMatrix.Items.Add("Матрица")
For i = 0 To n
    s = ""
```

Оглавление

```
    For j = 0 To n
        s += Str(a(i, j)) + vbTab
    Next
    lstMatrix.Items.Add(s)
Next
min = a(0, n)
imin = 0
jmin = n
max = a(n, n)
imax = n
jmax = n
For i = 0 To n
    For j = 0 To n
        If i < j And a(i, j) < min Then
            min = a(i, j)
            imin = i
            jmin = j
        End If
        If i + j > n And a(i, j) > max Then
            max = a(i, j)
            imax = i
            jmax = j
        End If
    Next
Next
lstMatrix.Items.Add("-----------------------------------")
lstMatrix.Items.Add("Минимальный над главной диагональю")
lstMatrix.Items.Add("Значение:" + Str(min))
lstMatrix.Items.Add("Строка:" + Str(imin))
lstMatrix.Items.Add("Столбец:" + Str(jmin))
lstMatrix.Items.Add("Максимальный под побочной диагональю")
lstMatrix.Items.Add("Значение:" + Str(max))
lstMatrix.Items.Add("Строка:" + Str(imax))
lstMatrix.Items.Add("Столбец:" + Str(jmax))
```

Оглавление

Список литературы

1. Волчёнков Н.Г. Программирование на Visual Basic 6: В 3-х ч. Часть 1. – М.: ИНФРА-М, 2000. – 286 с.
2. Шевякова Д.А., Степанов А.М., Карпов Р.Г. Самоучитель Visual Basic 2005 / под общ. ред. А.Ф. Тихонова. – СПб.: БХВ-Петербург, 2007. – 576 с.
3. Богданов М.Р. Visual Basic 2005 на примерах. – СПб.: БХВ-Петербург, 2007. – 592 с.

Оглавление