Оглавление

Оглавление

Введение

Массив – это множество однотипных данных, имеющих одинаковое имя и отличающихся друг от друга только порядковым номером, который называется индексом.

На практике массивы применяются для хранения и обработки больших объемов однотипных данных. Но эти данные доступны только во время выполнения программы. После завершения работы программы данные, хранящиеся в массивах, теряются.

Для обработки массивов используются специальные алгоритмы. Наиболее распространенными из них являются сортировка массива, поиск максимального и минимального элементов массива, формирование нового массива из элементов исходного массива, поиск одного или нескольких элементов, удовлетворяющих некоторому заранее заданному условию. Мы подробно рассмотрим большинство основных алгоритмов, но сначала сформулируем основное правило обработки одномерных массивов.

Массивы всегда обрабатываются в цикле.

Причем счетчик цикла используется в качестве индекса массива.

В зависимости от количества индексов массивы делятся на одномерные (с одним индексом) и многомерные (с двумя и более индексами).

Одномерный массив – это средство языка программирования, которое позволяет обращаться к любому элементу пронумерованного множества значений. При этом все элементы множества должны иметь одинаковый тип данных, а их количество не должно превосходить некоторого заранее заданного числа, которое называется размером массива.

1. Объявление массива

Объявление массива выполняется аналогично объявлению переменной. Для этого используются уже знакомые нам операторы `Dim`, `Static`, `Public` и `Private`. По способу описания массивы делятся на две основные группы[1]:

- массив с заранее известным числом элементов;

[1] В предыдущих версиях Visual Basic массивы по способу описания делились на статические и динамические. Размер статического массива определялся один раз при его описании и не мог меняться в процессе выполнения программы. Размер динамического массива при описании не указывался, а задавался в процессе выполнения программы. Размер динамического массива можно было многократно менять в процессе выполнения программы. Причем допускалось как уменьшение размера массива, так и его увеличение. Начиная с версии Visual Basic 2005, понятие статического массива исчезает. Все массивы в Visual Basic 2005 являются динамическими, независимо от способа их объявления.

Оглавление

- массив, число элементов которого заранее неизвестно.

При описании массива указывается его имя, затем ставятся круглые скобки, показывающие Visual Basic 2005, что мы организуем массив, и задается тип данных, к которому будут принадлежать все элементы массива. Рассмотрим два случая объявления массива.

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

Эта конструкция описывает одномерный целочисленный массив `a` типа `Integer`, размер которого заранее неизвестен. Изначально в нем нет ни одного элемента. Размер этого массива будет определен позднее с помощью оператора `ReDim`.

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

Такая запись определяет одномерный массив `b` типа `Single`, в котором будет содержаться 11 элементов пронумерованных от 0 до 10. То есть при описании массива в круглых скобках указывается номер его последнего элемента (это значение не должно выходить за пределы значений типа `ULong`). Обратите внимание, что нумерация элементов массива всегда начинается с нуля независимо от способа объявления массива. Размер этого массива тоже можно будет изменять в процессе выполнения программы с помощью оператора `ReDim`.

Оператор `ReDim` предназначен для изменения размера массива в процессе выполнения программы. Причем размер массива можно как уменьшать, так и увеличивать. Но он не позволяет изменить тип элементов массива и размерность массива (количество используемых индексов). При изменении размера массива данные, хранящиеся в нем, могут теряться. Чтобы этого не происходило после слова `ReDim` необходимо поставить ключевое слово `Preserve`. В общем виде оператор ReDim записывается следующим образом

```
ReDim Имя массива(Номер последнего элемента)
```

или

```
ReDim Preserve Имя массива(Номер последнего элемента)
```

Рассмотрим два примера использования оператора `ReDim`. Запись

```
ReDim a(6)
```

изменяет размер массива `a`. После выполнения этого оператора в массиве будет 7 элементов, пронумерованных от 0 до 6, но все значения, ранее записанные в массив `a`, потеряются.

Конструкция

Оглавление

```
ReDim Preserve b(7)
```

тоже приведет к изменению размера массива `b`. В нем станет 8 элементов, пронумерованных от 0 до 7, но при этом сохранятся все значения, которые вводились в массив `b`.

Объем памяти, необходимой для хранения массива, определяется как произведение количества элементов массива и объема памяти, занимаемого одной переменной указанного типа данных.

Для того чтобы обратиться к некоторому элементу массива, необходимо указать имя массива и в круглых скобках номер нужного элемента. Например:

```
a(3) = 5
b(5) = (a(0) + a(4)) / b(1)
```

2. Ввод массива

Ввести массив – значит задать его размер и указать значения всех его элементов. Существует два основных способа заполнения массива. В первом случае значения элементов последовательно вводятся с клавиатуры. Во втором случае они задаются с помощью генератора случайных чисел. Разберем оба 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.

Сначала описывается целочисленный массив `a()`. Так как его размер заранее неизвестен, то массив описывается без указания верхней границы.

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

Для работы с массивом нам необходимо знать его размер. Он будет храниться в переменной `n`. Поскольку массивы всегда обрабатываются в цикле, то для организации цикла `For` нам потребуется счетчик `i`. Очевидно, что обе эти переменных всегда будут иметь целый тип.

```
Dim n, i As Integer
```

Задание массива начинается с определения его размера. Мы просим пользователя указать количество элементов в массиве. Так как количество элементов может быть только положительным, то при вводе этого значения необходима проверка, которую мы организуем с помощью цикла `Do Loop Until`.

Оглавление

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

В Visual Basic 2005 нумерация элементов массива всегда начинается с нуля. Следовательно, номер последнего элемента будет на единицу меньше общего количества элементов массива. Поэтому уменьшаем значение переменной `n` на единицу. Теперь в ней хранится не количество элементов, а номер последнего элемента массива.

```
n -= 1
```

Задаем размер массива `a()`, указывая в операторе `ReDim` номер последнего элемента массива.

```
ReDim a(n)
```

Организуем цикл для ввода значений элементов массива. Элементы массива последовательно пронумерованы от 0 до `n`. Следовательно, счетчик цикла должен изменяться в этом же диапазоне. Тогда на `i`-м шаге цикла мы будем вводить элемент массива с номером `i`.

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

С помощью функции `InputBox` вводим значение `i`-го элемента массива. Так как вводимое значение является числом, то используем преобразование `Val`.

```
    a(i) = Val(InputBox("Введите " + Str(i) + _
                        "-й элемент массива"))
Next
```

Второй способ ввода массива – это заполнение случайными числами. В Visual Basic 2005 есть специальная функция, которая по определенным правилам генерирует рациональные случайные числа в диапазоне [0; 1). Она называется Rnd(). Используя эту функцию, можно заполнить массив случайными числами из любого диапазона. Рассмотрим задачу заполнения целочисленного массива случайными числами из некоторого диапазона. Начало и конец диапазона значений задается с клавиатуры.

Сначала описывается целочисленный массив `a()`. Так как его размер заранее неизвестен, то массив описывается без указания верхней границы.

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

Для работы с массивом нам необходимо знать его размер. Он будет храниться в переменной `n`. Поскольку массивы всегда обрабатываются в цикле, то для организации

Оглавление

цикла For нам потребуется счетчик i. Очевидно, что обе эти переменных всегда будут иметь целый тип.

```
Dim n, i As Integer
```

Также нам потребуются переменные для хранения начала и конца диапазона случайных чисел. Заведем соответствующие переменные start (начало диапазона) и fin (конец диапазона).

```
Dim start, fin As Integer
```

Задание массива начинается с определения его размера. Мы просим пользователя указать количество элементов в массиве. Так как количество элементов может быть только положительным, то при вводе этого значения необходима проверка, которую мы организуем с помощью цикла Do Loop Until.

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

В Visual Basic 2005 нумерация элементов массива всегда начинается с нуля. Следовательно, номер последнего элемента будет на единицу меньше общего количества элементов массива. Поэтому уменьшаем значение переменной n на единицу. Теперь в ней хранится не количество элементов, а номер последнего элемента массива.

```
n -= 1
```

Задаем размер массива a(), указывая в операторе ReDim номер последнего элемента массива.

```
ReDim a(n)
```

Вводим начало и конец диапазона случайных чисел.

```
start = Val(InputBox("Введите начало отрезка"))
fin = Val(InputBox("Введите конец отрезка"))
```

Выполняем начальную настройку генератора случайных чисел. Для этого используется функция Randomize().

```
Randomize()
```

Организуем цикл для ввода значений элементов массива. Элементы массива последовательно пронумерованы от 0 до n. Следовательно, счетчик цикла должен изменяться в этом же диапазоне. Тогда на i-м шаге цикла мы будем вводить элемент массива с номером i.

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

Оглавление

На каждом шаге цикла вычисляем значение очередного элемента массива. Функция `Rnd()` возвращает рациональное случайное число в диапазоне [0; 1). Умножив это значение на разность между концом и началом желаемого диапазона, получим диапазон [0; `fin - start`). Так как в общем случае начало диапазона отличается от нуля, то полученный диапазон случайных чисел надо сдвинуть в требуемую точку. Для этого добавим к нашему числу величину равную началу диапазона. Получим [0 + `start`; `fin – start + start`) или [`start`; `fin`). Так как заполняемый массив является целочисленным, то полученное значение необходимо округлить. Функция `Math.Round` реализует процесс математического округления. В итоге мы получаем целое случайное число из диапазона [`start`; `fin`]. Описанный процесс реализуется следующим арифметическим выражением.

```
        a(i) = Math.Round(start + (fin - start) * Rnd())
    Next
```

Полный текст этой программы представлен в приложении 1. Пример работы программы приведен на рис. 1. Исходные данные для этого случая: `n` = 10, `start` = 10, `fin` = 10.

3. Вывод массива в окно списка и в текстовое поле

Вывод массива – одна из основных операций, без которой не обходится ни одна программа обработки массивов. Массивы можно выводить в окно списка или в текстовое поле. Рассмотрим оба способа.

Вывод массива в окно списка практически не отличается от вывода совокупности. Сначала очищаем окно списка.

```
    lstA.Items.Clear()
```

Выводим в окно списка поясняющий текст и заголовки колонок. Константа `vbTab` позволяет организовать вывод в две колонки.

```
    lstA.Items.Add("Исходный массив")
    lstA.Items.Add("Номер" + vbTab + "Значение")
```

Организуем цикл для вывода значений элементов массива. Элементы массива последовательно пронумерованы от 0 до `n`. Следовательно, счетчик цикла должен изменяться в этом же диапазоне. Тогда на `i`-м шаге цикла мы будем выводить элемент массива с номером `i`.

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

На каждом шаге цикла в окно списка выводим номер элемента – `i` и его значение – `a(i)`. Константа `vbTab` позволяет организовать вывод в две колонки.

Оглавление

```
            lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
        Next
```

Вывод массива в текстовое поле существенно отличается от вывода в окно списка. Здесь основным приемом является своеобразное накопление суммы. Другое отличие заключается в организации вывода. Если в окно списка мы выводили каждый элемент массива по отдельности, то в текстовое поле выводится сразу весь массив. Сначала в специальной переменной формируется выводимая информация, а выводится она только после завершения основного цикла. Рассмотрим особенности программной реализации этого процесса.

Для вывода массива в текстовое поле нам потребуется специальная переменная, в которой будет формироваться выводимая строка. Так как массив выводится в текстовое поле, то эта переменная будет иметь символьный тип данных, а точнее – `String`.

```
        Dim s As String
```

Сначала в этой строке нет никакой информации. Она пустая. Пустая строка обозначается парой кавычек, между которыми нет ни одного символа.

```
        s = ""
```

Организуем цикл для формирования строки выводимой информации. Элементы массива последовательно пронумерованы от 0 до `n`. Следовательно, счетчик цикла должен изменяться в этом же диапазоне. Тогда на `i`-м шаге цикла мы будем обрабатывать элемент массива с номером `i`.

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

На каждом шаге цикла мы будем добавлять к переменной `s` очередной элемент массива, который предварительно преобразуем в символьный формат с помощью функции `Str`. Чтобы между элементами массива на экране было некоторое расстояние, после каждого элемента будем добавлять к строке `s` один пробел, который необходимо взять в кавычки. Так как для строк операция сложения имеет смыл соединения (склейки) строк, то в строке `s` постепенно будут скапливаться все элементы массива, разделенные пробелом.

```
            s += Str(a(i)) + " "
        Next
```

После завершения основного цикла в переменной `s` будут записаны все элементы массива. Остается только вывести значение переменной `s` в текстовое поле. Так как

Оглавление

переменная s имеет символьный тип данных, то при ее выводе не надо использовать никаких дополнительных преобразований.

```
txtA.Text = s
```

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

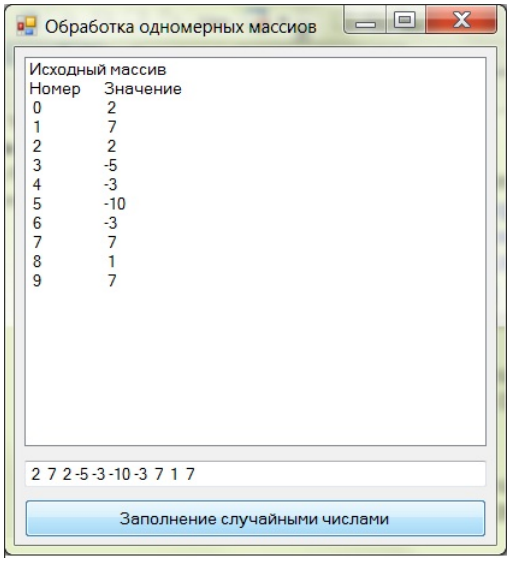

Рис. 1. Заполнение массива случайными числами и вывод его в окно списка и в текстовое поле

4. Вычисление суммы и произведения элементов массива

Вычисление суммы и произведения всех элементов массива практически не отличается от аналогичных операций для обработки совокупности чисел. Вычисление происходит путем последовательного накопления значения. Начальное значение суммы – ноль, произведения – единица. На каждом шаге цикла к ранее накопленному значению суммы добавляется значение очередного элемента массива, а значение произведения умножается на значение текущего элемента массива. Рассмотрим особенности программной реализации данного алгоритма.

Объявляем переменные для суммы и произведения соответственно. Тип этих переменных должен всегда совпадать с типом элементов массива.

```
Dim summa, proiz As Integer
```

Задаем начальные значения для суммы и произведения.

```
summa = 0
proiz = 1
```

Оглавление

Организуем цикл для формирования суммы и произведения. Элементы массива последовательно пронумерованы от 0 до `n`. Следовательно, счетчик цикла должен изменяться в этом же диапазоне. Тогда на `i`-м шаге цикла мы будем обрабатывать элемент массива с номером `i`.

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

Добавляем очередной элемент массива к общей сумме.

```
    summa += a(i)
```

Значение произведения умножаем на значение текущего элемента массива.

```
    proiz *= a(i)
Next
```

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

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

Выводим значение накопленной суммы и поясняющий текст.

```
lstA.Items.Add("Сумма = " + Str(summa))
```

Выводим значение произведения и поясняющий текст.

```
lstA.Items.Add("Произведение = " + Str(proiz))
```

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

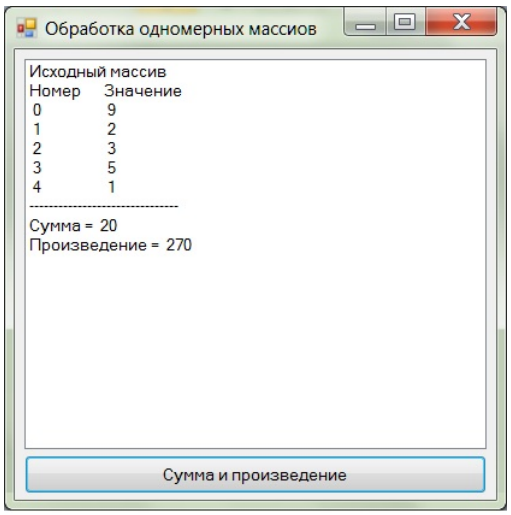

Рис. 2. Пример работы программы вычисления суммы и произведения всех элементов массива

Оглавление

5. Определение количества элементов массива, удовлетворяющих некоторому условию

Определение количества элементов массива, удовлетворяющих некоторому условию, выполняется практически также как и при обработке совокупности чисел. До начала обработки массива количество нужных элементов полагается равным нулю. На каждом шаге цикла проверяем, соответствует ли данный элемент поставленному условию. Если значение элемента удовлетворяет условию, то искомое количество увеличивается на единицу. Рассмотрим особенности программной реализации данного алгоритма на примере задачи определения количества положительных элементов в массиве.

Объявляем переменную для хранения результатов вычислений. Так как количество элементов – всегда является целым числом, то эта переменная будет иметь тип `Integer`.

```
Dim kol As Integer
```

До начала анализа полагаем количество искомых элементов равным нулю.

```
kol = 0
```

Организуем цикл для определения количества положительных элементов. Элементы массива последовательно пронумерованы от 0 до `n`. Следовательно, счетчик цикла должен изменяться в этом же диапазоне. Тогда на `i`-м шаге цикла мы будем обрабатывать элемент массива с номером `i`.

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

На каждом шаге цикла проверяем, является ли текущий элемент массива положительным.

```
    If a(i) > 0 Then
```

Если да, то количество положительных элементов увеличивается на единицу.

```
        kol += 1
    End If
Next
```

После завершения основного цикла выводим полученные результаты. Сначала выводим горизонтальную черту, чтобы зрительно отделить результаты от исходных данных.

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

Анализируем полученное значение.

```
If kol = 0 Then
```

Оглавление

Если количество положительных элементов равно нулю, значит, в массиве нет ни одного положительного элемента. В этом случае вместо ответа выводим поясняющий текст.

```
            lstA.Items.Add("Нет положительных элементов")
        Else
```

В противном случае выводим найденное количество положительных элементов.

```
            lstA.Items.Add("Количество положительных =" + _
                           Str(kol))
        End If
```

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

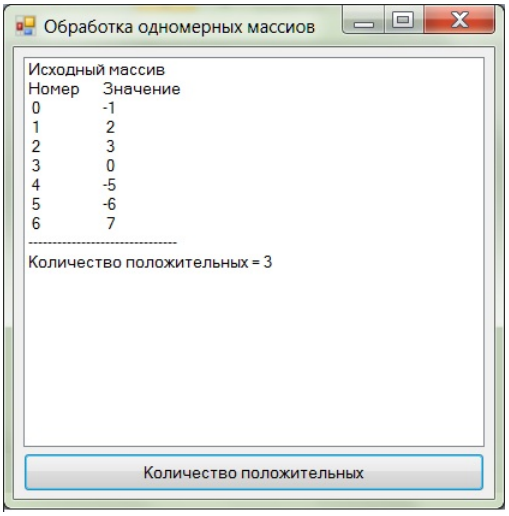

Рис. 3. Пример работы программы определения количества положительных элементов в массиве

6. Вычисление среднего арифметического и среднего геометрического элементов массива, удовлетворяющих некоторому условию

Алгоритмы решения задач вычисления среднего арифметического и среднего геометрического элементов массива, удовлетворяющих некоторому условию, представляют собой комбинацию алгоритмов определения количества элементов массива, удовлетворяющих некоторому условию, и вычисления суммы и произведения элементов массива соответственно. Для вычисления среднего арифметического необходимо найти сумму и количество элементов, удовлетворяющих поставленному условию. А для вычисления среднего геометрического надо найти произведение и количество нужных элементов (определение среднего геометрического нескольких

Оглавление

чисел приведено в разделе 6.1.3). Рассмотрим особенности программной реализации этих алгоритмов на примере задачи вычисления среднего арифметического четных элементов и среднего геометрического нечетных элементов массива.

Объявляем переменные, необходимые для решения задачи. Переменные для хранения суммы и произведения всегда будут иметь тот же тип данных, что и элементы массива.

```
Dim summa, proiz As Integer
```

Переменные для хранения количества четных (`kol1`) и нечетных (`kol2`) элементов массива, очевидно, будут иметь целый тип.

```
Dim kol1, kol2 As Integer
```

Среднее арифметическое получается в результате деления, поэтому оно всегда имеет рациональный тип. То же относится и к среднему геометрическому, которое получается в результате извлечения корня некоторой степени.

```
Dim arifm, geom As Single
```

Задаем начальные значения: для суммы – ноль, для произведения – единица.

```
summa = 0
proiz = 1
```

До начала анализа элементов массива оба количества полагаются равными нулю.

```
kol1 = 0
kol2 = 0
```

Организуем цикл для анализа элементов массива. Элементы массива последовательно пронумерованы от 0 до `n`. Следовательно, счетчик цикла должен изменяться в этом же диапазоне. Тогда на `i`-м шаге цикла мы будем обрабатывать элемент массива с номером `i`.

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

На каждом шаге проверяем, является ли очередной элемент массива четным числом. Четные числа делятся на два без остатка. Другими словами, для четных чисел остаток при делении на два равен нулю.

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

Если текущий элемент массива является четным числом, то увеличиваем на единицу количество четных элементов массива, а к сумме добавляем значение элемента массива.

```
        kol1 += 1
        summa += a(i)
```

Оглавление

```
            Else
```

В противном случае, если элемент массива является нечетным числом, мы увеличиваем на единицу количество нечетных чисел, а накопленное произведение умножаем на значение текущего элемента массива.

```
                kol2 += 1
                proiz *= a(i)
            End If
        Next
```

После окончания основного цикла анализируем полученные результаты, вычисляем значения среднего арифметического и среднего геометрического, если это возможно, и выводим ответы в окно списка. Чтобы зрительно отделить исходные данные от результатов вычислений, первым делом выведем горизонтальную черту.

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

Анализируем количество четных чисел.

```
        If kol1 = 0 Then
```

Если количество четных чисел равно нулю, значит, в массиве нет ни одного четного числа. В этом случае невозможно вычислить их среднее арифметическое. Поэтому вместо ответа надо вывести поясняющий текст.

```
            lstA.Items.Add("Нет четных")
        Else
```

В противном случае мы вычисляем среднее арифметическое.

```
            arifm = summa / kol1
```

Полученный результат мы выводим в окно списка.

```
            lstA.Items.Add("Сред. арифм. четных = " + _
                           Str(arifm))
        End If
```

Теперь анализируем количество нечетных чисел.

```
        If kol2 = 0 Then
```

Если количество нечетных чисел равно нулю, значит, в массиве нет ни одного нечетного числа. В этом случае невозможно вычислить их среднее арифметическое. Поэтому вместо ответа надо вывести поясняющий текст.

```
            lstA.Items.Add("Нет нечетных")
        Else
```

Иначе мы анализируем знак подкоренного выражения.

```
            If proiz > 0 Then
```

Оглавление

Если произведение нечетных чисел положительно, то мы можем вычислить их среднее геометрическое без дополнительных преобразований.

```
geom = proiz ^ (1 / kol2)
```

Полученное значение выводим в окно списка.

```
lstA.Items.Add("Сред. геом. нечетных = " + _
                    Str(geom))
        Else
```

В противном случае (если подкоренное выражение отрицательное) мы должны проверить четность степени корня.

```
If kol2 Mod 2 = 0 Then
```

Если требуется извлечь корень четной степени, то задача вычисления среднего геометрического не имеет решения, так как извлечение корня четной степени из отрицательного числа невозможно. Поэтому вместо ответа выводим поясняющий текст.

```
lstA.Items.Add("Невозможно " + _
                    " вычислить сред. геом.")
        Else
```

Иначе, если степень корня нечетная, то для вычисления корня потребуется составить арифметическое выражение. В Visual Basic 2005 операция извлечения корня произвольной степени определена только для положительных подкоренных выражений. Поэтому, когда необходимо извлечь корень нечетной степени из отрицательного числа, поступают следующим образом. Корень извлекается из модуля подкоренного выражения, а у полученного результата знак меняется на противоположный.

```
geom = -Math.Abs(proiz) ^ (1 / kol2)
```

Полученный результат выводим в окно списка.

```
lstA.Items.Add("Сред. геом. " + _
                    "нечетных = " + Str(geom))
        End If
    End If
End If
```

Полный текст программы представлен в приложении 4. Примеры работы программы приведены на рис. 4.

Оглавление

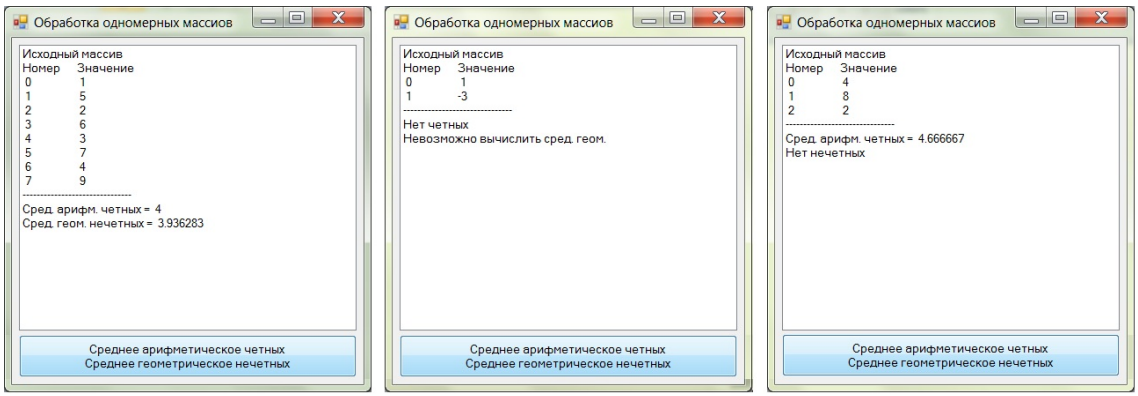

Рис. 4. Примеры работы программы вычисления среднего арифметического четных и среднего геометрического нечетных элементов массива

7. Нахождение максимального элемента массива

Для поиска максимального элемента массива и определения его номера нам потребуются две дополнительные переменные. В первой из них будет храниться само максимальное значение, а во второй – его индекс. Поиск максимального элемента в массиве традиционно начинают с элемента, имеющего номер ноль. Затем организуют цикл, в котором анализируют все элементы массива, кроме начального. Если значение какой-либо элемента массива окажется больше ранее найденного максимума, то значение максимума необходимо обновить, сделав равным этому элементу. Сразу же происходит и запоминание номера элемента массива, в котором найдено новое максимальное значение. 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 массива. В переменной `imax` мы будем запоминать номер максимального элемента массива. Эта переменная всегда будет иметь целочисленный тип.

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

Поиск максимального элемента начинает с элемента, имеющего номер ноль. Записываем значение этого элемента в переменную `max`.

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

Соответственно в переменную `imax` мы записываем номер этого элемента, то есть ноль.

```
imax = 0
```

Так как сравнение нулевого элемента с самим собой не даст нам никакой новой информации, то основной цикл мы должны начать не с нулевого элемента, а с первого.

Оглавление

```
For i = 1 To n
```

На каждом шаге цикла анализируем значение очередного элемента массива.

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

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

```
max = a(i)
```

Сразу же запоминаем индекс этого элемента массива.

```
imax = i
End If
Next
```

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

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

Выводим значение максимального элемента.

```
lstA.Items.Add("Максимальное = " + Str(max))
```

Выводим номер максимального элемента.

```
lstA.Items.Add("Его номер = " + Str(imax))
```

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

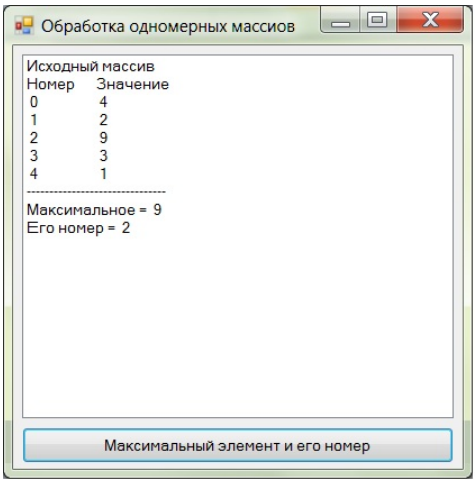

Рис. 5. Пример работы программы нахождения максимального элемента массива и его номера

Оглавление

8. Нахождение минимального элемента массива, удовлетворяющего некоторому условию

При внешней схожести с задачей, рассмотренной в предыдущем разделе, задача поиска минимального элемента массива, удовлетворяющего некоторому условию (например, кратного трем), решается несколько иначе. Так как значения элементов массива заранее неизвестны, то мы не можем начать поиск с какого-то конкретного элемента. Более того, возможна ситуация, когда в исходном массиве не будет ни одного элемента, удовлетворяющего поставленному условию. Поэтому поиск нужного элемента и его номера организуется также как и при обработке совокупности чисел. Начальное значение минимума полагается равным некоторому достаточно большому числу. Начальное значение номера берется за пределами массива, например, -1. Элементы массива анализируются подряд, начиная с нулевого. Рассмотрим особенности программной реализации алгоритма поиска минимального элемента массива, кратного трем, и его номера.

Объявляем необходимые переменные. Переменная `min` предназначена для хранения значения минимального элемента, кратного трем. Ее тип всегда должен совпадать с типом элементов массива. В переменной `imin` мы будем запоминать номер найденного элемента массива. Эта переменная всегда будет иметь целочисленный тип.

```
Dim min, imin As Integer
```

Так как в процессе обработки элементов массива значение минимума будет уменьшаться, то начальное значение для переменной `min` надо задать достаточно большим.

```
min = 100000
```

Начальное значение переменной imin зададим равным -1 (то есть за пределами массива), чтобы показать, что мы пока не нашли ни одного элемента, удовлетворяющего поставленному условию.

```
imin = -1
```

Организуем цикл для обработки всех элементов массива. Цикл должен начать свою работу с нулевого элемента массива. Поэтому начальное значение счетчика – ноль.

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

На каждом шаге цикла проверяем значение очередного элемента массива.

```
If a(i) Mod 3 = 0 And a(i) < min Then
```

Оглавление

Если значение элемента массива без остатка делится на 3, значит, этот элемент кратен трем. Если при этом он меньше ранее найденного минимума, то значение минимума необходимо обновить, записав в него значение текущего элемента массива.

```
                min = a(i)
```

Сразу же запоминаем номер этого элемента массива.

```
                imin = i
            End If
        Next
```

После завершения цикла нам необходимо вывести полученные результаты в окно списка. Сначала выводим горизонтальную строку, чтобы зрительно отделить результаты от исходных данных.

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

Анализируем значение переменной `imin`.

```
        If imin = -1 Then
```

Если после цикла значение переменной `imin` осталось равным -1, значит, в массиве нет ни одного элемента, кратного трем. Поэтому вместо ответа выводим поясняющий текст.

```
            lstA.Items.Add("Нет чисел, кратных 3")
        Else
```

В противном случае выводим найденные значения: минимальный элемент массива, кратный трем, и его номер.

```
            lstA.Items.Add("Минимальное кратное трем =" + _
                           Str(min))
             lstA.Items.Add("Его номер = " + Str(imin))
        End If
```

Полный текст программы представлен в приложении 6. Примеры работы программы при различных исходных данных приведены на рис. 6.

Оглавление

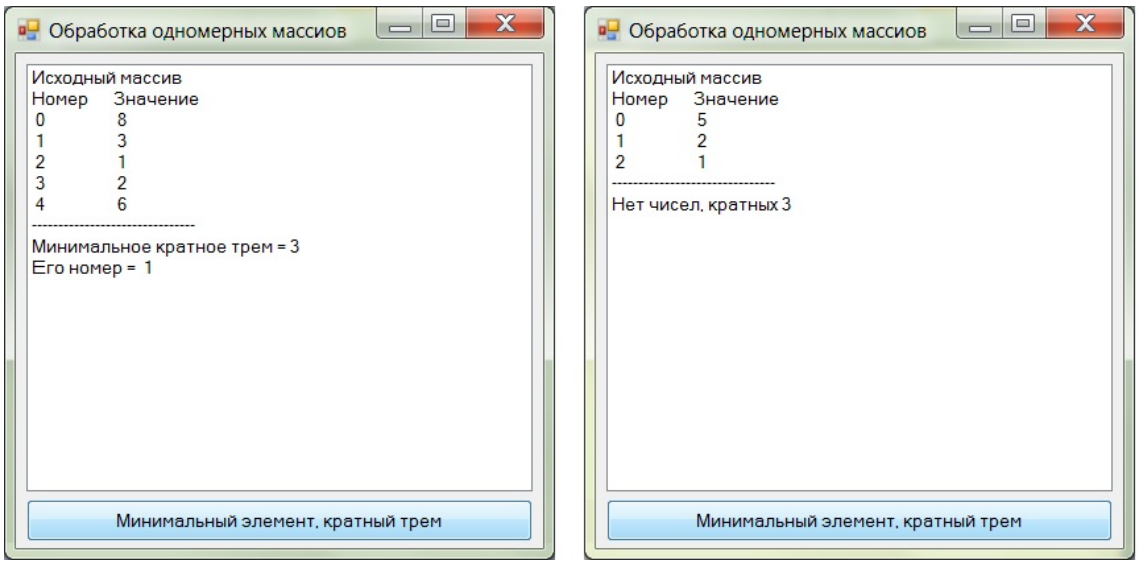

Рис. 6. Примеры работы программы поиска минимального элемента массива, кратного трем, и его номера

9. Поиск первого элемента массива, удовлетворяющего некоторому условию

При решении задача поиска первого элемента массива, удовлетворяющего некоторому условию, предполагается, что таких элементов в массиве несколько. Найти первый из подходящих элементов, значит, найти его номер. Считается, что поиск идет слева направо, от нулевого элемента к последнему элементу. Но при этом нельзя исключать ситуацию, когда в массиве вообще нет искомых элементов.

Сначала предполагают, что в массиве нет искомого элемента. Поэтому его номер задается равным -1. Затем последовательно проверяются все элементы массива. Если какой-нибудь элемент удовлетворяет поставленному условию, его номер запоминается в специальной переменной, а сам цикл немедленно прерывается. Так как по условию задачи требуется найти только первый элемент, удовлетворяющий условию, то после его обнаружения анализ других элементов массива является бессмысленным. Рассмотрим особенности реализации данного алгоритма на примере задачи поиска номера первого элемента массива, равного нулю.

Объявляем переменную, в которой будет храниться номер искомого элемента. Очевидно, что эта переменная всегда будет иметь целочисленный тип данных.

```
Dim perv As Integer
```

Задаем начальное значение этой переменной. Так как мы предполагаем, что нужных элементов в массиве нет, то начальное значение возьмем равным -1.

```
perv = -1
```

Оглавление

Организуем цикл для анализа всех элементов массива, от нулевого до последнего, который имеет номер n.

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

Анализируем очередной элемент массива.

```
    If a(i) = 0 Then
```

Если он равен нулю, значит, мы нашли первый из подходящих элементов. Запоминаем его номер в специальной переменной.

```
        perv = i
```

И прерываем цикл, так как продолжать поиск не требуется.

```
        Exit For
    End If
Next
```

После завершения цикла выводим полученные результаты в окно списка. Сначала выводим горизонтальную черту, чтобы зрительно отделить ответы от исходных данных.

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

Затем анализируем полученный результат.

```
If perv = -1 Then
```

Если номер первого искомого элемента равен -1, значит, в массиве не было ни одного элемента, удовлетворяющего поставленному условию. В этом случае вместо ответа надо вывести поясняющий текст.

```
    lstA.Items.Add("В массиве нет нулей")
Else
```

В противном случае мы выводим найденное значение.

```
    lstA.Items.Add("Первый нулевой элемент: " + _
                   Str(perv))
End If
```

Полный текст программы представлен в приложении 7. Примеры работы программы при различных исходных данных приведены на рис. 7.

Оглавление

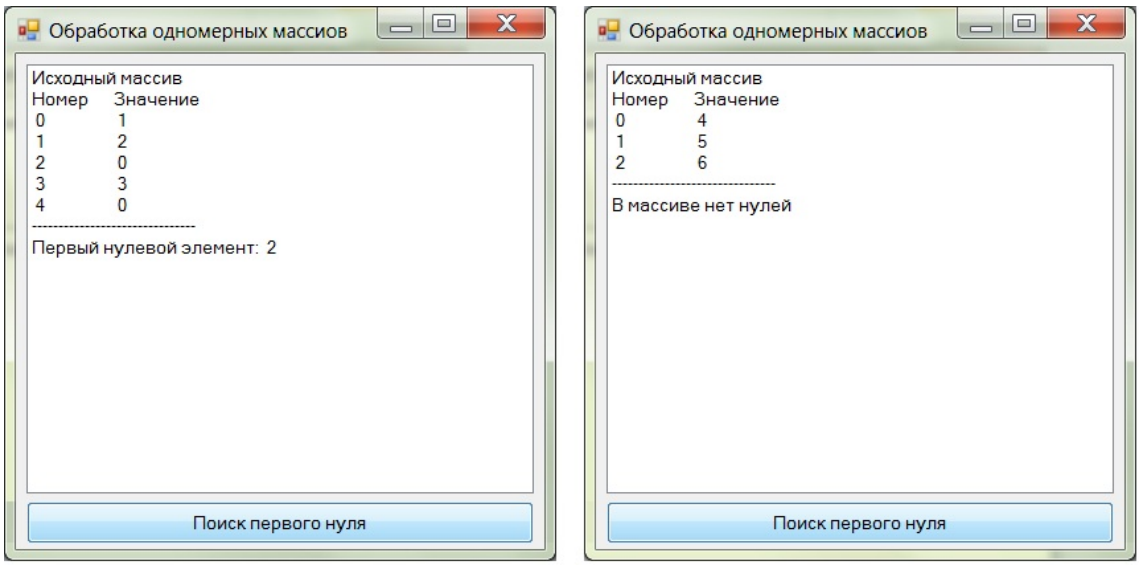

Рис. 7. Пример работы программы поиска первого нулевого элемента

10. Поиск последнего элемента массива, удовлетворяющего некоторому условию

Задача поиска последнего элемента, удовлетворяющего некоторому условию, имеет два разных решения.

Первый способ предлагает анализировать массив от нулевого элемента к последнему. Как только в нем встречается очередной элемент, удовлетворяющий поставленному условию, его номер запоминается в специальной переменной. Но в отличие от предыдущей задачи цикл не прерывается, а продолжается дальше. Так как все номера последовательно записываются в одну и ту же переменную, то после завершения цикла в ней будет храниться номер последнего элемента, удовлетворяющего заданному условию.

Второй способ позволяет свести данную задачу к предыдущей. Очевидно, если анализировать массив в обратном направлении, то есть от последнего элемента к нулевому (от конца к началу), то первый же элемент, удовлетворяющий поставленному условию, и будет искомым элементом. Достаточно запомнить его номер и прервать цикл.

Рассмотрим особенности реализации обоих алгоритмов на примере решения задачи поиска последнего элемента массива, равного нулю.

Первый способ.

Объявляем переменную, в которой будет храниться номер искомого элемента. Очевидно, что эта переменная всегда будет иметь целочисленный тип данных.

```
Dim posled As Integer
```

Оглавление

Задаем начальное значение этой переменной. Так как мы предполагаем, что нужных элементов в массиве нет, то начальное значение возьмем равным -1.

```
posled = -1
```

Организуем цикл для анализа всех элементов массива, от нулевого до последнего, который имеет номер n.

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

Анализируем текущий элемент массива.

```
    If a(i) = 0 Then
```

Если он равен нулю, значит, мы нашли очередной подходящий элемент. Запоминаем его номер в специальной переменной.

```
        posled = i
    End If
Next
```

После завершения цикла выводим полученные результаты в окно списка. Сначала выводим горизонтальную черту, чтобы зрительно отделить ответы от исходных данных.

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

Затем анализируем полученный результат.

```
If posled = -1 Then
```

Если номер искомого элемента равен -1, значит, в массиве не было ни одного элемента, удовлетворяющего поставленному условию. В этом случае вместо ответа надо вывести поясняющий текст.

```
    lstA.Items.Add("В массиве нет нулей")
Else
```

В противном случае мы выводим найденное значение.

```
lstA.Items.Add("Последний нулевой элемент: " + _
                    Str(posled))
End If
```

Второй способ.

Объявляем переменную, в которой будет храниться номер искомого элемента. Очевидно, что эта переменная всегда будет иметь целочисленный тип данных.

```
Dim posled As Integer
```

Задаем начальное значение этой переменной. Так как мы предполагаем, что нужных элементов в массиве нет, то начальное значение возьмем равным -1.

Оглавление

```
posled = -1
```

Организуем цикл для анализа всех элементов массива, от последнего, имеющего номер n, до нулевого. Так как движение по массиву идет в обратную сторону, то мы должны задать шаг изменения счетчика. На каждом шаге цикла номер очередного элемента массива будет уменьшаться на единицу. Поэтому шаг цикла надо задать равным -1. Для этого используется ключевое слово Step.

```
For i = n To 0 Step -1
```

Анализируем очередной элемент массива.

```
    If a(i) = 0 Then
```

Если он равен нулю, значит, мы нашли подходящий элемент. Запоминаем его номер в специальной переменной.

```
        posled = i
```

И прерываем цикл, так как продолжать поиск не требуется.

```
        Exit For
    End If
Next
```

После завершения цикла выводим полученные результаты в окно списка. Сначала выводим горизонтальную черту, чтобы зрительно отделить ответы от исходных данных.

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

Затем анализируем полученный результат.

```
If posled = -1 Then
```

Если номер искомого элемента равен -1, значит, в массиве не было ни одного элемента, удовлетворяющего поставленному условию. В этом случае вместо ответа надо вывести поясняющий текст.

```
    lstA.Items.Add("В массиве нет нулей")
 Else
```

В противном случае мы выводим найденное значение.

```
    lstA.Items.Add("Последний нулевой элемент: " + _
                   Str(posled))
End If
```

Полный текст программы представлен в приложении 8. Примеры работы программы при различных исходных данных приведены на рис. 8.

Оглавление

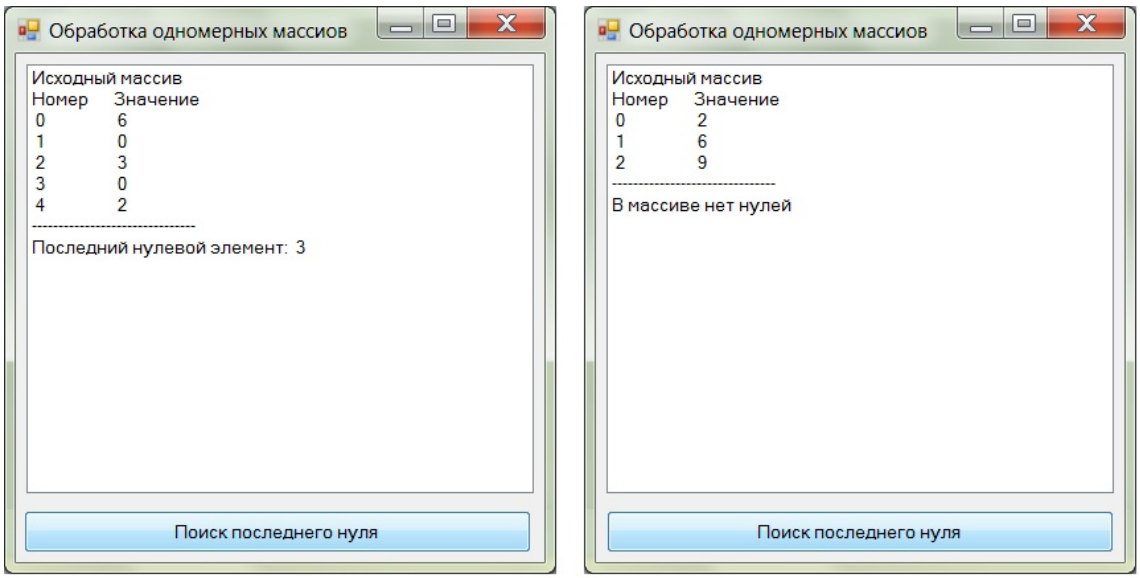

Рис. 8. Примеры работы программы поиска последнего нулевого элемента

11. Замена одного элемента массива

Для замены одного элемента массива на некоторое значение необходимо знать номер заменяемого элемента и само значение. Следовательно, данная задача будет решаться за несколько шагов. Сначала необходимо вычислить нужное значение. Затем, если при вычислении не возникли ошибки, и не было выхода за пределы области допустимых значений, надо определить номер заменяемого элемента. Обычно первые два шага алгоритма стараются объединить в одном цикле. На третьем заключительном шаге алгоритма выполняется непосредственная замена значения выбранного элемента массива. При решении этой задачи массив необходимо выводить дважды. Первый раз элементы массива печатаются непосредственно после ввода. Это исходное состояние массива. Второй раз элементы массива выводятся уже после замены элементов. Это отображается измененный (преобразованный) массив. В результате пользователь может увидеть, как изменился массив в результате работы программы.

Рассмотрим особенности программной реализации данного алгоритма на примере задачи замены максимального элемента массива на сумму всех элементов массива.

Для решения задачи нам потребуются следующие переменные: `max` – значение максимального элемента, `imax` – номер максимального элемента, `summa` – сумма всех элементов массива.

```
Dim max, imax, summa As Integer
```

Задаем начальные значения для поиска максимального элемента и его номера.

Оглавление

```
max = a(0)
imax = 0
```

Задаем начальное значение для суммы. Мы собираемся объединить в одном цикле поиск максимума и вычисление суммы, но при этом возникает противоречие. Вычисляя сумму всех элементов массива, мы начинали обрабатывать массив с нулевого элемента. А поиск максимального элемента традиционно начинается с первого элемента массива. Чтобы разрешить это противоречие, мы зададим начальное значение суммы равным нулевому элементу, а обработку массива начнем с первого элемента.

```
summa = a(0)
```

Организуем цикл для обработки элементов массива.

```
For i = 1 To n
```

Добавляем очередной элемент массива к общей сумме, накапливая таким образом итоговое значение.

```
    summa += a(i)
```

Анализируем текущий элемент массива.

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

Если его значение превышает значение максимума, значит, максимум надо обновить, записав в него значение этого элемента.

```
        max = a(i)
```

Запоминаем новый номер максимального элемента.

```
        imax = i
    End If
Next
```

После завершения цикла выполняем замену. В массив на место элемента, номер которого хранится в переменной `imax`, то есть на место максимума, записываем значение вычисленной суммы.

```
a(imax) = summa
```

Теперь можно выводить результаты. Сначала выводим горизонтальную черту, чтобы зрительно отделить исходные данные от результатов работы программы.

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

Выводим поясняющий заголовок.

```
lstA.Items.Add("Измененный массив")
```

Затем последовательно в цикле печатаем все элементы массива. Использование константы `vbTab` позволяет организовать вывод в две колонки.

Оглавление

```
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
```

Полный текст программы представлен в приложении 9. Пример работы программы приведен на рис. 9.

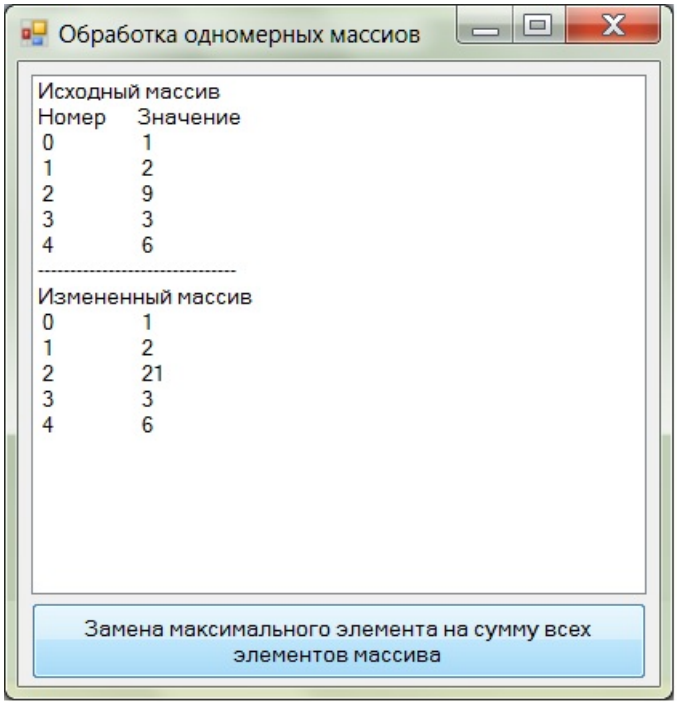

Рис. 9. Пример работы программы, заменяющей максимальный элемент массива на сумму всех его элементов

12. Замена всех элементов массива, удовлетворяющих некоторому условию

Эта задача во многом похожа на предыдущую. Сначала необходимо вычислить нужное значение, а затем заменить на него все элементы массива, удовлетворяющие поставленному условию. Единственное отличие заключается в том, что замена происходит в цикле, так как в массиве может быть несколько элементов, подлежащих замене. При этом на каждом шаге анализируется очередной элемент массива. Если он удовлетворяет заданному условию, то происходит его замена. Очевидно, что массив необходимо выводить дважды. До и после преобразования, чтобы пользователь мог проконтролировать правильность работы программы.

Рассмотрим особенности программой реализации этого алгоритма на примере задачи замены всех элементов, равных нулю, на -1.

Оглавление

Начнем с объявления необходимых переменных. Так как в массиве может не быть ни одного элемента, равного нулю, то нам потребуется переменная, в которой будет храниться количество выполненных замен. Очевидно, что эта переменная будет иметь целочисленный тип данных.

```
Dim kol As Integer
```

До начала преобразования количество выполненных замен полагаем равным нулю, так как еще не было выполнено ни одной замены.

```
kol = 0
```

Организуем цикл для обработки всех элементов исходного массива.

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

На каждом шаге цикла анализируем очередной элемент массива.

```
    If a(i) = 0 Then
```

Если этот элемент равен нулю, значит, его необходимо заменить на -1.

```
        a(i) = -1
```

Соответственно количество выполненных замен увеличивается на единицу.

```
        kol += 1
    End If
Next
```

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

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

Затем анализируем количество выполненных замен.

```
If kol = 0 Then
```

Ели это количество равно нулю, значит, в массиве не было ни одного элемента, равного нулю. Следовательно, исходный массив не изменился, и выводить его не надо. Поэтому вместо преобразованного массива мы выводим поясняющий текст.

```
    lstA.Items.Add("В массиве нет нулей")
Else
```

В противном случае, если замены были, выводим поясняющий заголовок.

```
    lstA.Items.Add("Измененный массив")
```

Затем последовательно в цикле печатаем все элементы измененного массива. Использование константы `vbTab` позволяет организовать вывод в две колонки.

```
    For i = 0 To n
        lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
```

Оглавление

```
        Next
    End If
```

Полный текст программы представлен в приложении 10. Примеры работы программы при различных исходных данных приведены на рис. 10.

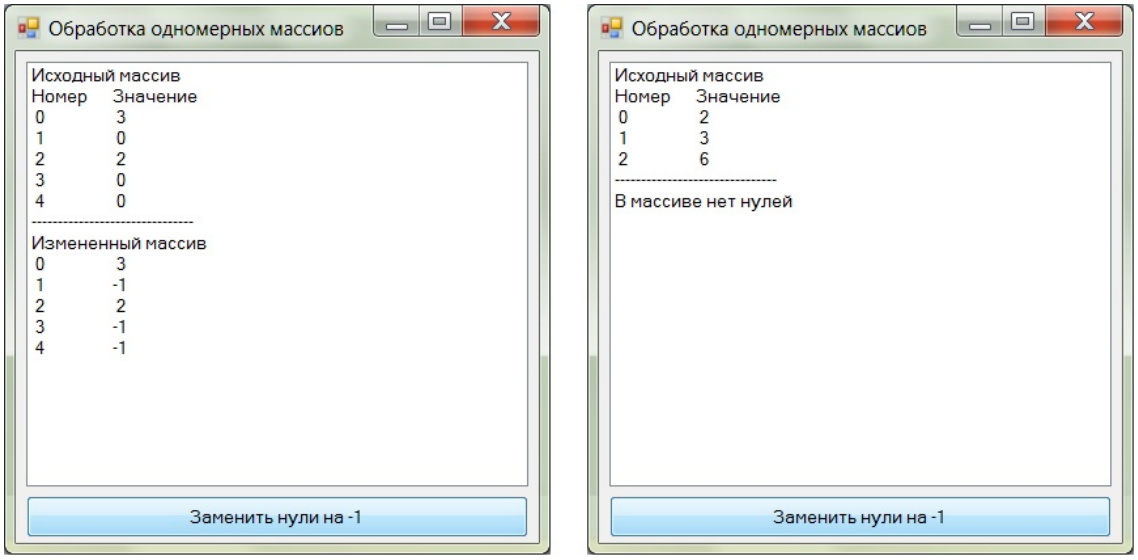

Рис. 10. Примеры работы программы, заменяющей все нулевые элементы массива на -1

13. Перестановка местами двух элементов массива

Задача перестановки местами двух элементов массива решается тем же способом, что и задача обмена значений двух переменных. Для перестановки местами двух элементов массива нам необходимо знать номера этих элементов. Перед перестановкой необходимо проверить, существуют ли в массиве элементы с такими номерами, не выходят ли эти номера за границы массива. Если перестановка возможна, то для ее проведения нам потребуется дополнительная переменная, в которой временно будет храниться значение одного из переставляемых элементов. Очевидно, что массив надо будет выводить дважды: до и после перестановки элементов, чтобы пользователь мог проконтролировать правильность работы программы.

Рассмотрим особенности реализации этого алгоритма. Номера переставляемых элементов будем вводить с клавиатуры, используя функцию `InputBox`.

Начинаем с объявления переменных. Номера переставляемых элементов будем хранить в переменных k и p, которые будут иметь целочисленный тип данных.

```
Dim k, p As Integer
```

Оглавление

Для перестановки элементов массива нам потребуется дополнительная переменная, в которой будет храниться значение одного из переставляемых элементов. Тип этой переменной всегда будет совпадать с типом элементов массива.

```
Dim z As Integer
```

Вводим номер первого переставляемого элемента. При вводе нам необходимо организовать проверку, чтобы пользователь не мог ввести номер, находящиеся за пределами массива. Для этого достаточно потребовать, чтобы вводимое значение находилось в диапазоне от 0 до n включительно, где n – номер последнего элемента массива.

```
Do
   k = Val(InputBox("Введите номер элемента <=" + _
                    Str(n)))
Loop Until k >= 0 And k <= n
```

Аналогичным образом вводим номер второго переставляемого элемента.

```
Do
   p = Val(InputBox("Введите номер элемента <=" + _
                    Str(n)))
Loop Until p >= 0 And p <= n
```

В дополнительной переменной сохраняем значение первого из переставляемых элементов.

```
z = a(k)
```

На место этого элемента записываем значение второго переставляемого элемента массива.

```
a(k) = a(p)
```

А на место второго элемента записываем старое значение первого элемента, сохраненное в дополнительной переменной.

```
a(p) = z
```

В результате элементы массива поменялись местами. Теперь измененный массив необходимо вывести в окно списка. Сначала выводим горизонтальную черту, чтобы зрительно отделить исходные данные от результатов.

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

Печатаем поясняющий заголовок.

```
lstA.Items.Add("Измененный массив")
```

Оглавление

Затем последовательно в цикле печатаем все элементы измененного массива. Использование константы vbTab позволяет организовать вывод в две колонки.

```
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
```

Полный текст программы представлен в приложении 11. Пример работы программы приведен на рис. 11.

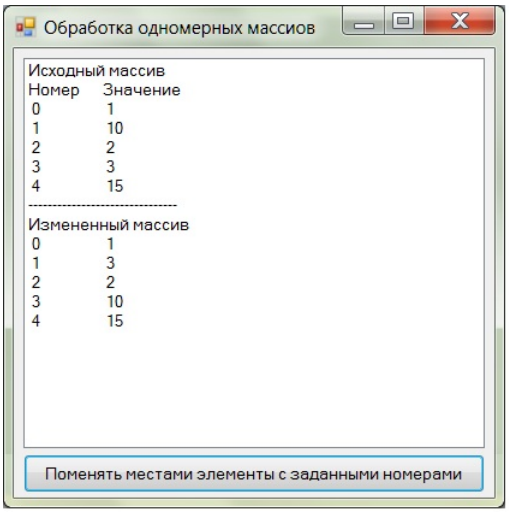

Рис. 11. Пример работы программы, переставляющей местами два элемента массива

14. Формирование нового массива из некоторых элементов исходного массива

При решении этой задачи в новый массив переносятся те элементы исходного массива, которые удовлетворяют оставленному условию. Формируемый массив описывается как массив с неизвестным числом элементов. На каждом шаге основного цикла анализируется очередной элемент исходного массива. Если он удовлетворят заданному условию, то размер формируемого массива увеличивается на единицу, и на место последнего элемента записывается проверяемый элемент исходного массива. После завершения основного цикла сформированный массив необходимо вывести, чтобы пользователь мог проконтролировать правильность работы программы. Рассмотрим особенности программной реализации данного алгоритма на примере задачи формирования нового массива из четных элементов исходного массива.

Объявляем новый массив. Так как количество элементов исходного массива, удовлетворяющих заданному условию, заранее неизвестно, то при описании нового

Оглавление

массива его размер не указывается. Очевидно, что тип элементов нового массива всегда будет совпадать с типом элементов исходного массива.

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

Так как в новый массив переносятся только те элементы исходного массива, которые удовлетворяют заданному условию, то количество элементов в новом массиве, скорее всего, будет отличаться от количества элементов в исходном массиве. Соответственно поменяются и номера элементов. Поэтому нам потребуется переменная для хранения номера последнего элемента в новом массиве. Она будет иметь целочисленный тип.

```
Dim k As Integer
```

До начала формирования нового массива в нем не содержится ни одного элемента. Поэтому номер последнего элемента будет находиться за границами массива. Как правило, его задают равным -1.

```
k = -1
```

Организуем цикл для проверки всех элементов исходного массива.

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

Анализируем очередной элемент исходного массива

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

Если он имеет четное значение, то его необходимо записать в новый массив. При этом на единицу увеличится количество элементов в новом массиве. Следовательно, на единицу увеличится и номер его последнего элемента.

```
            k += 1
```

Изменяем размер нового массива с помощью оператора `ReDim`. Чтобы при изменении размера массива не потерялись ранее найденные элементы, необходимо использовать ключевое слово `Preserve`.

```
            ReDim Preserve b(k)
```

В новый массив на место последнего элемента записываем анализируемый элемент исходного массива.

```
            b(k) = a(i)
        End If
Next
```

После завершения основного цикла в массиве `b()` находятся все четные элементы исходного массива. Остается распечатать все элементы нового массива в окне списка. Выводим горизонтальную черту, чтобы зрительно отделить исходные данные от результатов.

Оглавление

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

Анализируем номер последнего элемента в новом массиве.

```
If k = -1 Then
```

Если он остался равным -1, значит, в исходном массиве не было ни одного элемента, удовлетворяющего поставленному условию. Следовательно, в новом массиве нет ни одного элемента. Поэтому вместо элементов массива выведем поясняющее сообщение.

```
    lstA.Items.Add("Новый массив пуст")
Else
```

В противном случае новый массив содержит хотя бы один элемент, и мы можем организовать вывод массива. Сначала печатаем поясняющий заголовок.

```
    lstA.Items.Add("Новый массив")
```

Затем последовательно в цикле печатаем все элементы сформированного массива. Обратите внимание, что при выводе массива `b()` используется переменная `k` (а не `n`), то есть номер последнего элемента в новом массиве. Константа `vbTab` позволяет организовать вывод в две колонки.

```
    For i = 0 To k
        lstA.Items.Add(Str(i) + vbTab + Str(b(i)))
    Next
End If
```

Полный текст программы представлен в приложении 12. Примеры работы программы при различных исходных данных приведены на рис. 12.

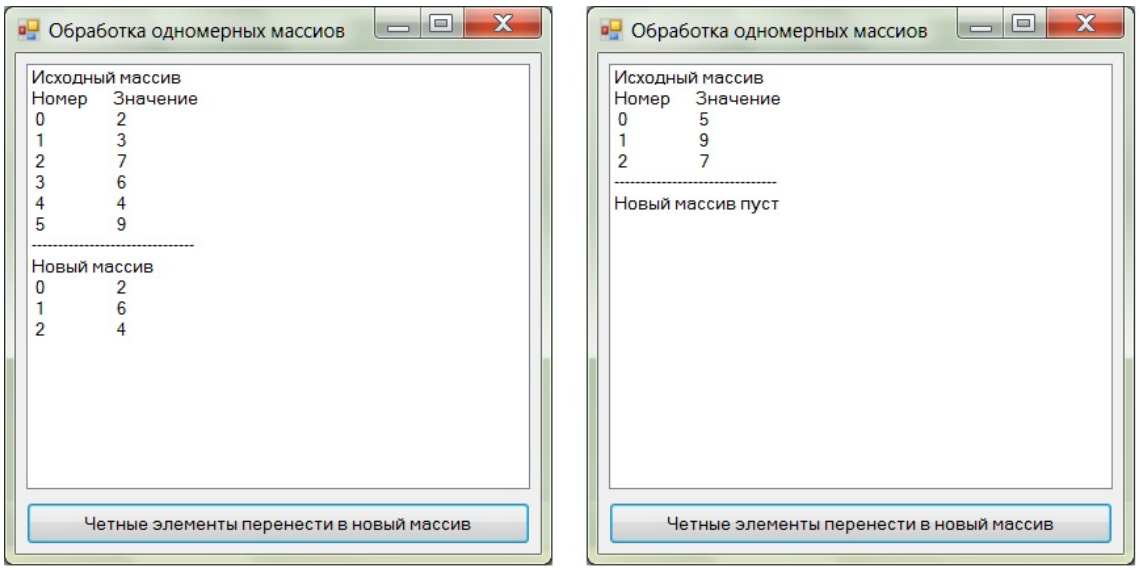

Рис. 12. Примеры работы программы формирования нового массива из четных элементов исходного массива

Оглавление

15. Проверка совпадения всех элементов массива

Задача проверки совпадения всех элементов массива решается двумя разными способами. Они оба используют тот факт, что если все элементы массива одинаковые, то они обязательно будут совпадать с нулевым элементом массива.

Первый способ предлагает определить, если ли в массиве хотя бы один элемент, отличающийся от нулевого. Если да, значит, в не все элементы массива совпадают. Для этого последовательно анализируются все элементы массива. Если какой-нибудь элемент отличается от нулевого, то делается вывод, что в массиве есть различные элементы и цикл прерывается.

Второй способ предлагает определить количество элементов, совпадающих с нулевым. Если это количество совпадет с общим количеством элементов в массиве, значит, массив состоит из одинаковых элементов.

Рассмотрим особенности программной реализации обоих алгоритмов.

Первый способ.

Для решения задач нам потребуется логическая переменная, в которой будет храниться результат проверки массива.

```
Dim test As Boolean
```

Изначально предполагаем, что все элементы в массиве одинаковые. Тогда результатом проверки будет Истина (`True`).

```
test = True
```

Анализируем все элементы исходного массива, кроме нулевого. Очевидно, что нулевой элемент всегда будет совпадать с самим собой. Поэтому анализ элементов массива начинаем с элемента, имеющего номер 1.

```
For i = 1 To n
```

Сравниваем очередной элемент массива с нулевым.

```
    If a(i) <> a(0) Then
```

Если они не совпадают, значит, в массиве есть различные элементы. Поэтому в результирующую переменную записываем значение Ложь (`False`).

```
            test = False
```

И прерываем выполнение цикла, так как решение задачи уже получено.

```
            Exit For
    End If
Next
```

Оглавление

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

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

Затем анализируем полученный ответ. Обратите внимание на формулировку логического выражения в условном операторе. Оно записано в виде логической переменной[2]. При выполнении программы вместо логической переменной будет подставлено ее значение. Если оно будет равно Истине (`True`), то выполнятся операторы, стоящие в части `Then`. Если логическая переменная `test` будет иметь значение Ложь (`False`), то выполнятся операторы, стоящие в части `Else`.

```
If test Then
```

Если в переменной `test` записано значение Истина (`True`), значит, все элементы массива одинаковые. Выводим соответствующее сообщение.

```
    lstA.Items.Add("Все элементы одинаковые")
Else
```

В противном случае, если в переменной `test` хранится значение Ложь (`False`), значит, в массиве есть различные элементы.

```
    lstA.Items.Add("Есть несовпадающие элементы")
End If
```

Второй способ.

В этом способ вместо логической переменной используется целочисленная переменная, в которой хранится количество элементов массива, равных нулевому элементу.

```
Dim kol As Integer
```

До начала анализа количество совпадений полагаем равным нулю.

```
kol = 0
```

Организуем цикл для проверки все элементов массива.

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

Анализируем текущий элемент массива.

```
    If a(i) = a(0) Then
```

Если он совпадает с нулевым элементом массива, то увеличиваем количество совпадений на единицу.

---

[2] Visual Basic 2005 позволяет сформулировать условное выражение в более привычном виде.

```
If test = True Then ...
```

Работоспособность программы при этом не изменится, но такая запись считается в программировании «дурным тоном».

Оглавление

```
                kol += 1
            End If
        Next
```

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

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

Затем анализируем найденное количество совпадений с нулевым элементом массива.

```
        If kol = n + 1 Then
```

Если полученное значение на единицу превышает номер последнего элемента массива (то есть равно количеству элементов), значит, все элементы массива совпадают с нулевым. Следовательно, массив состоит из одинаковых элементов.

```
            lstA.Items.Add("Все элементы одинаковые")
        Else
```

В противном случае в массиве есть несовпадающие элементы. И мы выводим соответствующее сообщение.

```
            lstA.Items.Add("Есть несовпадающие элементы")
        End If
```

Полный текст программы представлен в приложении 13. Примеры работы программы при различных исходных данных приведены на рис. 13.

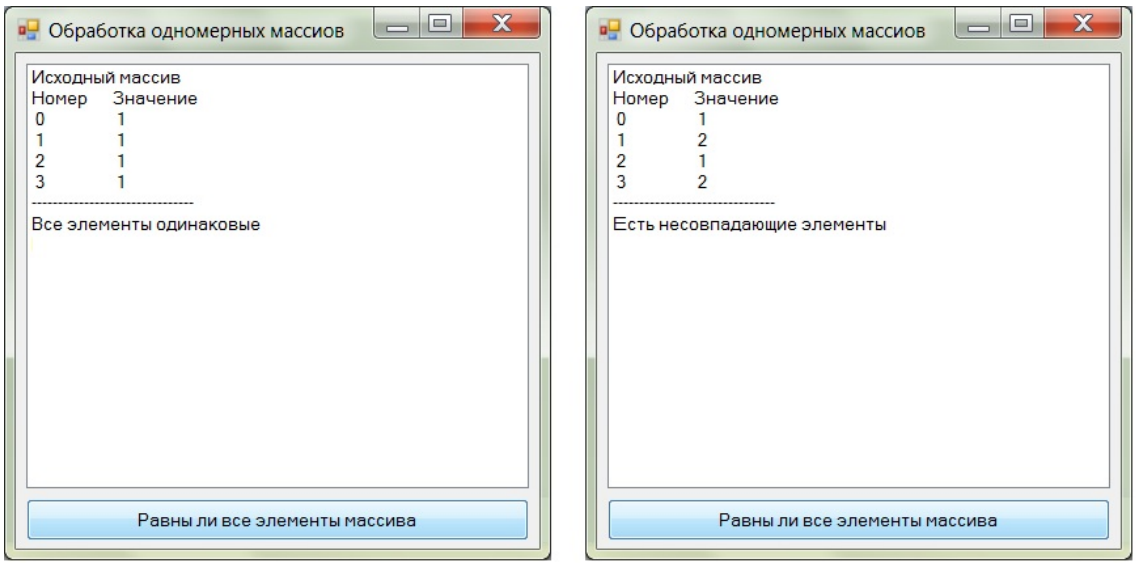

Рис. 13. Примеры работы программы, проверяющей, совпадают ли все элементы массива

Оглавление

16. Проверка упорядоченности всех элементов массива

Массив называется упорядоченным (отсортированным) по возрастанию, если каждый элемент массива не меньше всех предшествующих элементов и не больше всех последующих элементов. Другим словами, для любых двух элементов массива, отсортированного по возрастанию, выполняется условие.

Если `i < j`, то `a(i) <= a(j)`,

где `i` и `j` – номера элементов массива, `a(i)` и `a(j)` – элементы массива с соответствующими номерами.

Массив называется упорядоченным (отсортированным) по убыванию, если каждый элемент массива не больше всех предшествующих элементов и не меньше всех последующих элементов. Другим словами, для любых двух элементов массива, отсортированного по убыванию, выполняется условие.

Если `i < j`, то `a(i) >= a(j)`,

где `i` и `j` – номера элементов массива, `a(i)` и `a(j)` – элементы массива с соответствующими номерами.

Для проверки упорядоченности массива необходимо последовательно сравнить все соседние элементы массива. Если в массиве найдется хотя бы одна пара элементов, для которой не выполняется условие упорядоченности, значит, массив не отсортирован.

Рассмотрим особенности программной реализации данного алгоритма. Требуется проверить, упорядочены ли элементы целочисленного массива по возрастанию.

Для решения задачи нам потребуется логическая переменная, в которой будет храниться результат программы.

```
Dim test As Boolean
```

До начала проверки предполагаем, что массив упорядочен. Следовательно, в результирующую переменную мы записываем значение Истина (`True`).

```
test = True
```

Организуем цикл для проверки всех элементов массива, от нулевого до предпоследнего. Так как на каждом шаге цикла мы будем сравнивать элементы с номерами `i` и `(i+1)`, то завершить цикл надо, когда значение счетчика `i` станет равным `n-1`. В этом случае на последнем шаге цикла будут сравниваться элементы с номерами `(n-1)` и `n`. Если цикл после этого шага не прервать, то программа станет сравнивать элементы с номерами `n` и `(n+1)`. Элемент с номером `(n+1)` в массиве

Оглавление

отсутствует, поэтому произойдет аварийное завершении программы и ответ получен не будет.

```
For i = 0 To n – 1
```

Анализируем соседние элементы.

```
    If a(i) > a(i + 1) Then
```

Если элемент `a(i)` больше элемента `a(i + 1)`, то есть предшествующий элемент больше, чем последующий, значит, массив не отсортирован по возрастанию. Записываем в результирующую переменную значение Ложь (`False`).

```
        test = False
```

Теперь выполнение цикла можно прекратить, так как ответ уже получен.

```
        Exit For
    End If
Next
```

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

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

Затем анализируем полученный ответ. Обратите внимание на формулировку логического выражения в условном операторе. Она такая же, как в разделе 7.14.

```
If test Then
```

Если в переменной `test` записано значение Истина (`True`), значит, элементы массива упорядочены. Выводим соответствующее сообщение.

```
    lstA.Items.Add("Элементы массива упорядочены")
Else
```

В противном случае, если в переменной `test` хранится значение Ложь (`False`), значит, элементы массива неупорядочены.

```
    lstA.Items.Add("Элементы массива неупорядочены")
End If
```

Полный текст программы представлен в приложении 14. Примеры работы программы при различных исходных данных приведены на рис. 14.

Оглавление

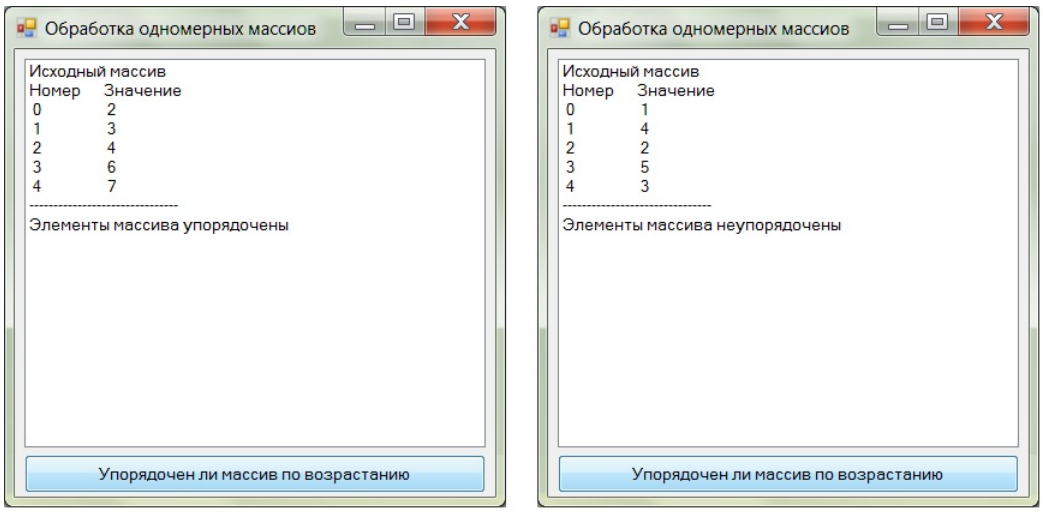

Рис. 14. Примеры работы программы, проверяющей, упорядочены ли элементы массива по возрастанию

17. Сортировка массива методом пузырька

Отсортировать массив – значит, переставить его элементы таким образом, чтобы для каждой пары выполнялось заданное условие упорядоченности (см. раздел 16). Существует множество различных способов сортировки массива. Мы рассмотрим два самый простых: метод пузырька (или метод наивной сортировки) и метод линейной сортировки (см. раздел 18).

Метод пузырька предлагает сравнивать каждый элемент с соседним. Если два элемента стоят неправильно, нарушая условие сортировки, то их меняют местами. Процесс перестановки продолжается до тех пор, пока все элементы не окажутся на своих местах. Тогда для всех пар элементов массива будет выполняться условие упорядоченности, и массив будет отсортирован. В таблице 1 приведен пример сортировки массива по возрастанию методом пузырька.

Таблица 1

Массив	Действие
1 3 5 2 4	Сравниваем первую пару элементов. Числа 1 и 3 стоят в правильном порядке. Их переставлять не надо. Делаем один шаг по массиву и сравниваем числа 3 и 5. Их тоже не надо переставлять. Делаем еще один шаг и сравниваем числа 5 и 2. Они стоят в неправильном порядке, поэтому мы их меняем местами.

Оглавление

Массив	Действие
1 3 2 5 4	Сдвигаемся еще на один элемент и сравниваем числа 5 и 4. Они тоже стоят неправильно, и мы меняем их местами.
1 3 2 4 5	Мы дошли до конца массива. Но массив пока неупорядочен. Поэтому мы возвращаемся к началу массива и продолжаем сравнение. Числа 1 и 3 идут в правильном порядке, а вот числа 3 и 2 порядок нарушают, поэтому мы меняем их местами.
1 2 3 4 5	Пройдя еще один раз по всему массиву, мы убеждаемся, что все элементы стоят на своих местах. Значит, процесс сортировки массива можно завершить.

Рассмотрим особенности программной реализации данного алгоритма на примере задачи сортировки по возрастанию элементов целочисленного массива.

Для решения задачи объявим вспомогательные переменные. Логическая переменная `sort` предназначена для хранения информации о состоянии массива. Она показывает, отсортированы элементы массива или нет.

```
Dim sort As Boolean
```

Так как сортировка массива связана с перестановкой его элементов, то нам потребуется переменная для временного хранения одного из переставляемых элементов (см. раздел 13). Очевидно, что тип этой переменной всегда будет совпадать с типом элементов массива.

```
Dim z As Integer
```

Нам потребуется несколько раз проходить по массиву, ища пары элементов, стоящих в неправильном порядке. Для этого организуем цикл. Число повторов заранее неизвестно, следовательно, надо использовать цикл с условием. Нам придется обязательно пройти по массиву хотя бы один раз, чтобы удостовериться, что все элементы стоят на своих местах. Поэтому применим цикл с постусловием. Его выполнение завершится, когда массив будет полностью отсортирован.

```
Do
```

Перед началом каждого нового прохода по массиву предполагаем, что он отсортирован. В переменную `sort` записываем значение Истина (`True`).

```
    sort = True
```

Оглавление

Организуем цикл для проверки всех элементов массива, от нулевого до предпоследнего. Так как на каждом шаге цикла мы будем сравнивать элементы с номерами `i` и `(i+1)`, то завершить цикл надо, когда значение счетчика `i` станет равным `n-1`. В этом случае на последнем шаге цикла будут сравниваться элементы с номерами `(n-1)` и `n`. Если цикл после этого шага не прервать, то программа станет сравнивать элементы с номерами `n` и `(n+1)`. Элемент с номером `(n+1)` в массиве отсутствует, поэтому произойдет аварийное завершении программы и ответ получен не будет.

```
For i = 0 To n – 1
```

Анализируем соседние элементы.

```
If a(i) > a(i + 1) Then
```

Если элемент `a(i)` больше элемента `a(i + 1)`, то есть предшествующий элемент больше, чем последующий, значит, элементы стоят неправильном порядке и их необходимо поменять местами. Для этого используем дополнительную переменную `z`. Процесс перестановки элементов описан в разделе 13.

```
z = a(i)
a(i) = a(i + 1)
a(i + 1) = z
```

Так как элементы стояли в неправильном порядке, значит, массив не был отсортирован, поэтому в логическую переменную `sort` записываем значение Ложь (`False`).

```
sort = False
End If
Next
Loop Until sort
```

После завершения всех циклов мы можем вывести отсортированный массив. Сначала выводим горизонтальную черту, чтобы зрительно отделить исходные данные от результатов.

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

Печатаем поясняющий заголовок.

```
lstA.Items.Add("Массив после сортировки")
```

Затем последовательно в цикле выводим все элементы отсортированного массива. Использование константы `vbTab` позволяет организовать вывод в две колонки.

```
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
```

Оглавление

```
Next
```

Полный текст программы представлен в приложении 15. Пример работы программы приведен на рис. 15.

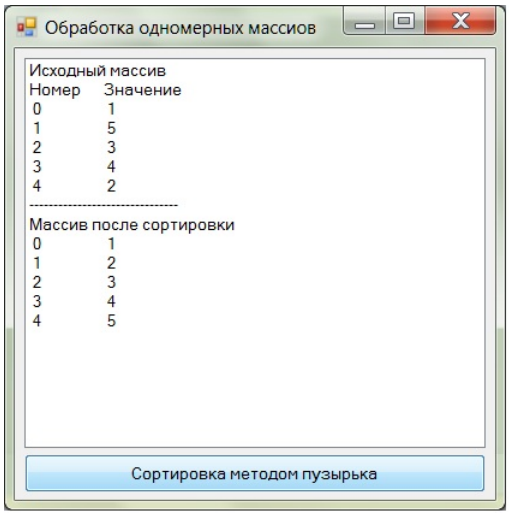

Рис. 15. Пример работы программы сортировки массива методом пузырька

18. Линейная сортировка массива (сортировка методом поиска минимума)

Сортировка массива методом поиска минимума является одним из частных случаев линейной сортировки. Она может использоваться как для сортировки по возрастанию, так и по убыванию.

При сортировке по возрастанию метод предлагает найти минимальный элемент в массиве и поставить его на нулевое место. А элемент с нулевого места переместить на место минимального элемента. После этого нулевой элемент массива гарантировано стоит на своем месте. Поэтому в дальнейшей сортировке он не участвует. На следующем шаге массив просматривается уже с первого элемента. В этой части находится свой минимум, которой меняется местами с первым элементом. Теперь уже два элемента массива стоят на своих местах. На следующем шаге массив обрабатывается со второго элемента. Процесс продолжается до тех пор, пока в необработанной части массива не останутся два элемента. Среди них тоже находится минимальный. Он ставится на предпоследнее место массива. А последний необработанный элемент автоматически попадает на последнее место в массиве. Теперь все элементы стоят на своих местах и процесс сортировки можно прекратить. В таблице 2 приведен пример сортировки массива по возрастанию методом поиска минимума.

Оглавление

Таблица 2

Массив	Действие
5 4 1 3 2	Обработку массива начинаем с нулевого элемента. Его значение равно 5. Находим минимальный элемент в массиве. Его значение равно 1, и он стоит на второй позиции. Меняем местами элементы с номерами 0 и 2.
1 4 5 3 2	Минимальный элемент гарантировано стоит на правильном месте, поэтому в дальнейшей сортировке он не участвует. Обработку массива начинаем с элемента под номером 1. Его значение равно 4. В оставшейся части массива ищем минимальный элемент. Его значение равно 2, и он стоит на четвертой позиции. Меняем местами элементы с номерами 1 и 4.
1 2 5 3 4	Теперь уже два элемента стоят на правильных местах. В дальнейшей сортировке они не участвуют. Обработку массива начинаем с элемента под номером 2. Его значение равно 5. В оставшейся части массива ищем минимальный элемент. Его значение равно 3, и он стоит на третьей позиции. Меняем местами элементы с номерами 2 и 3.
1 2 3 5 4	Уже три элемента стоят на правильных местах. В дальнейшей сортировке они не участвуют. В необработанной части массива осталось два элемента. Эта перестановка будет последней. Обработку массива начинаем с элемента под номером 3. Его значение равно 5. В оставшейся части массива ищем минимальный элемент. Его значение равно 4, и он стоит на четвертой позиции. Меняем местами элементы с номерами 3 и 4.
1 2 3 4 5	В результате последней перестановки оба необработанных элемента оказались на правильных местах. Процесс сортировки закончен.

Оглавление

Сортировка методом поиска минимума является одним из частных случаев линейной сортировки. Остальные варианты приведены в таблице 3.

Таблица 3

Направление сортировки	Метод сортировки	
	Метод поиска минимума	Метод поиска максимума
По возрастанию	В необработанной части массива ищется минимальный элемент, который потом меняется местами с первым элементом в необработанной части. Необработанная часть уменьшается слева на один элемент.	В необработанной части массива ищется максимальный элемент, который потом меняется местами с последним элементом в необработанной части. Необработанная часть уменьшается справа на один элемент.
По убыванию	В необработанной части массива ищется минимальный элемент, который потом меняется местами с последним элементом в необработанной части. Необработанная часть уменьшается справа на один элемент.	В необработанной части массива ищется максимальный элемент, который потом меняется местами с первым элементом в необработанной части. Необработанная часть уменьшается слева на один элемент.

Помимо этих вариантов существует еще один частный случай линейной сортировки. Это минимаксная сортировка (иногда встречается другое ее название – максиминная). Согласно данному методу в необработанной части массива одновременно ищется максимальный и минимальный элементы, которые затем расставляются по своим местам в зависимости от направления сортировки. При этом необработанная часть массива сокращается одновременно с двух сторон.

Рассмотрим особенности программной реализации алгоритма сортировки целочисленного массива по возрастанию методом поиска минимума.

Для решения задачи нам потребуется несколько дополнительных переменных. Переменная `start` предназначена для хранения номера элемента, с которого начинается необработанная часть массива.

Оглавление

```
Dim start As Integer
```

На каждом шаге основного цикла мы будем искать минимальный элемент в необработанной части массива. При этом нам потребуются переменные для хранения значения минимального элемента (`min`) и его номера (`imin`). Переменная `min` всегда будет иметь такой же тип, что и элементы массива. Переменная `imin` всегда будет иметь целочисленный тип данных.

```
Dim imin, min As Integer
```

Организуем основной цикл. На первом шаге алгоритма линейной сортировки начало необработанной части массива совпадает с началом самого массива, то есть с нулевым элементом. После каждого шага цикла начало необработанной части будет смещаться на один элемент к концу массива. На последнем аге цикла в необработанной части массива остается всего два элемента. Очевидно, что при этом начало необработанной части массива будет совпадать с предпоследним элементом массива, то есть с элементом, имеющим номер (`n-1`)

```
For start = 0 To n – 1
```

На каждом шаге цикла мы будем искать минимальный элемент в необработанной части массива. Поиск удобно начинать с первого элемента необработанной части. Его номер всегда совпадает со значением, записанным в переменной `start`. Запоминаем стартовый элемент необработанной части как минимум.

```
min = a(start)
```

В специальную переменную записываем соответствующий номер элемента.

```
imin = start
```

Организуем цикл для поиска минимального элемента в необработанной части массива. Цикл начнется с элемента, следующего за стартовым, и будет идти до конца необработанной части. В нашей задаче конец необработанной части совпадает с концом массива.

```
For i = start + 1 To n
```

Анализируем очередной элемент массива.

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

Если этот элемент меньше ранее найденного минимума, то значение минимума надо обновить, записав в него значение проверяемого элемента массива.

```
min = a(i)
```

Оглавление

В другую переменную записываем номер найденного минимального элемента. Этот номер мы будем использовать при перестановке элементов массива.

```
                imin = i
            End If
        Next
```

После завершения цикла поиска минимума, мы должны поменять местами найденное минимальное значение и стартовый элемент необработанной части массива. Это можно сделать двумя различными способами. Первый способ – традиционный, рассмотренный в разделе 13. Он предлагает использовать дополнительную переменную, в которую временно записывается значение одного из переставляемых элементов. Программный код при этом выглядит следующим образом.

```
        Dim z As Integer
        z = a(start)
        a(start) = a(imin)
        a(imin) = z
```

Второй способ применяется только при решении задачи линейной сортировки и не может быть распространен на другие случаи. На место минимального элемента из необработанной части записывается значение стартового элемента необработанной части.

```
        a(imin) = a(start)
```

Затем на место стартового элемента записывается значение минимального элемента, которое хранится в переменой `min`.

```
        a(start) = min
```

Никогда нельзя использовать одновременно оба способа перестановки элементов массива.

```
    Next
```

После завершения основного цикла выводим отсортированный массив. Сначала печатаем горизонтальную черту, чтобы зрительно отделить исходные данные от результатов.

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

Потом выводим поясняющий заголовок.

```
    lstA.Items.Add("Массив после сортировки")
```

И последовательно в цикле печатаем все элементы отсортированного массива. Использование константы `vbTab` позволяет организовать вывод в две колонки.

Оглавление

```
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
```

Полный текст программы представлен в приложении 16. Пример работы программы приведен на рис. 16.

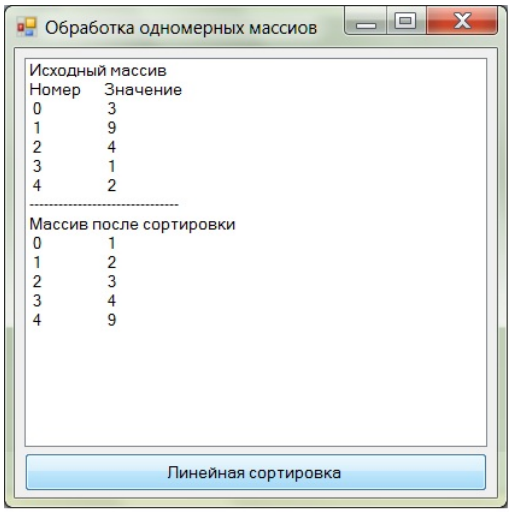

Рис. 51. Пример работы программы линейной сортировки массива

Приложение 1

Дан целочисленный массив. Заполнить его случайными числами из некоторого диапазона. Начало и конец диапазона значений задается с клавиатуры. Полученный массив вывести в окно списка и в текстовое поле.

```
Dim a() As Integer
Dim n, i As Integer
Dim start, fin As Integer
Dim s As String
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
start = Val(InputBox("Введите начало отрезка"))
```

Оглавление

```
fin = Val(InputBox("Введите конец отрезка"))
Randomize()
For i = 0 To n
    a(i) = Math.Round(start + (fin - start) * Rnd())
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
s = ""
For i = 0 To n
    s += Str(a(i)) + " "
Next
txtA.Text = s
```

Приложение 2

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Найти сумму и произведение всех элементов массива. Исходный массив и полученные результаты вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim summa, proiz As Integer
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
                "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
```

Оглавление

```
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
summa = 0
proiz = 1
For i = 0 To n
    summa += a(i)
    proiz *= a(i)
Next
lstA.Items.Add("------------------------------")
lstA.Items.Add("Сумма = " + Str(summa))
lstA.Items.Add("Произведение = " + Str(proiz))
```

Приложение 3

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Определить количество положительных элементов в массиве. Исходный массив и полученные результаты вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim kol As Integer
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
               "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
kol = 0
For i = 0 To n
```

Оглавление

```
    If a(i) > 0 Then
        kol += 1
    End If
Next
lstA.Items.Add("------------------------------")
If kol = 0 Then
    lstA.Items.Add("Нет положительных элементов")
Else
    lstA.Items.Add("Количество положительных =" + _
                   Str(kol))
End If
```

Приложение 4

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Вычислить среднее арифметическое четных элементов и среднее геометрическое нечетных элементов массива. Исходный массив и полученные результаты вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim summa, proiz As Integer
Dim kol1, kol2 As Integer
Dim arifm, geom As Single
lstA.Items.Clear()
Do
   n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
               "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
```

Оглавление

```
Next
summa = 0
proiz = 1
kol1 = 0
kol2 = 0
For i = 0 To n
    If a(i) Mod 2 = 0 Then
        kol1 += 1
        summa += a(i)
    Else
        kol2 += 1
        proiz *= a(i)
    End If
Next
lstA.Items.Add("------------------------------")
If kol1 = 0 Then
    lstA.Items.Add("Нет четных")
Else
    arifm = summa / kol1
    lstA.Items.Add("Сред. арифм. четных = " + _
                   Str(arifm))
End If
If kol2 = 0 Then
    lstA.Items.Add("Нет нечетных")
Else
    If proiz > 0 Then
        geom = proiz ^ (1 / kol2)
        lstA.Items.Add("Сред. геом. нечетных = " + _
                       Str(geom))
    Else
        If kol2 Mod 2 = 0 Then
            lstA.Items.Add("Невозможно " + _
                           " вычислить сред. геом.")
        Else
            geom = -Math.Abs(proiz) ^ (1 / kol2)
```

Оглавление

```
            lstA.Items.Add("Сред. геом. нечетных = " + _
                                Str(geom))
        End If
    End If
End If
```

Приложение 5

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Найти максимальный элемент массива и его номер. Исходный массив и полученные результаты вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim max, imax As Integer
lstA.Items.Clear()
Do
   n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
               "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
max = a(0)
imax = 0
For i = 1 To n
    If a(i) > max Then
       max = a(i)
       imax = i
    End If
Next
```

Оглавление

```
lstA.Items.Add("------------------------------")
lstA.Items.Add("Максимальное = " + Str(max))
lstA.Items.Add("Его номер = " + Str(imax))
```

Приложение 6

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Найти минимальный элемент массива, кратный трем, и его номер. Исходный массив и полученные результаты вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim min, imin As Integer
lstA.Items.Clear()
Do
   n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
                        "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
imin = -1
min = 100000
For i = 0 To n
    If a(i) Mod 3 = 0 And a(i) < min Then
        min = a(i)
        imin = i
    End If
Next
lstA.Items.Add("------------------------------")
If imin = -1 Then
```

Оглавление

```
    lstA.Items.Add("Нет чисел, кратных 3")
Else
    lstA.Items.Add("Минимальное кратное трем =" + _
                   Str(min))
     lstA.Items.Add("Его номер = " + Str(imin))
End If
```

Приложение 7

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Найти номер первого элемента массива, равного нулю. Исходный массив и полученные результаты вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim perv As Integer
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
               "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
perv = -1
For i = 0 To n
    If a(i) = 0 Then
        perv = i
        Exit For
    End If
Next
```

Оглавление

```
lstA.Items.Add("------------------------------")
If perv = -1 Then
    lstA.Items.Add("В массиве нет нулей")
Else
    lstA.Items.Add("Первый нулевой элемент: " + _
                   Str(perv))
End If
```

Приложение 8

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Найти номер последнего элемента массива, равного нулю. Исходный массив и полученные результаты вывести в окно списка.

Эта задача решается двумя разными способами.

Способ 1.

```
Dim a() As Integer
Dim n, i As Integer
Dim posled As Integer
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
               "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
posled = -1
For i = 0 To n
    If a(i) = 0 Then
        posled = i
```

Оглавление

```
    End If
Next
lstA.Items.Add("------------------------------")
If posled = -1 Then
    lstA.Items.Add("В массиве нет нулей")
Else
    lstA.Items.Add("Последний нулевой элемент: " + _
                   Str(posled))
End If
```

Способ 2.

```
Dim a() As Integer
Dim n, i As Integer
Dim posled As Integer
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
               "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
posled = -1
For i = n To 0 Step -1
    If a(i) = 0 Then
        posled = i
        Exit For
    End If
Next
```

Оглавление

```
lstA.Items.Add("------------------------------")
If posled = -1 Then
    lstA.Items.Add("В массиве нет нулей")
 Else
     lstA.Items.Add("Последний нулевой элемент: " + _
                    Str(posled))
End If
```

Приложение 9

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Заменить максимальный элемент массива на сумму всех элементов массива. Массив до и после преобразования вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim max, imax, summa As Integer
lstA.Items.Clear()
Do
   n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
                        "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
max = a(0)
imax = 0
summa = a(0)
For i = 1 To n
    summa += a(i)
    If a(i) > max Then
```

Оглавление

```
        max = a(i)
        imax = i
    End If
Next
a(imax) = summa
lstA.Items.Add("------------------------------")
lstA.Items.Add("Измененный массив")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
```

Приложение 10

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Заменить все элементы, равные нулю, на -1. Массив до и после преобразования вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim kol As Integer
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
                        "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
kol = 0
For i = 0 To n
    If a(i) = 0 Then
```

Оглавление

```
        a(i) = -1
        kol += 1
    End If
Next
lstA.Items.Add("------------------------------")
If kol = 0 Then
    lstA.Items.Add("В массиве нет нулей")
Else
    lstA.Items.Add("Измененный массив")
    For i = 0 To n
        lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
    Next
End If
```

Приложение 11

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Поменять местами элементы с заданными номерами. Номера переставляемых элементов задаются с клавиатуры. Массив до и после преобразования вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim k, p As Integer
Dim z As Integer
lstA.Items.Clear()
Do
   n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
                        "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
```

Оглавление

```
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
Do
    k = Val(InputBox("Введите номер элемента <=" + _
                     Str(n)))
Loop Until k >= 0 And k <= n
Do
    p = Val(InputBox("Введите номер элемента <=" + _
                     Str(n)))
Loop Until p >= 0 And p <= n
z = a(k)
a(k) = a(p)
a(p) = z
lstA.Items.Add("------------------------------")
lstA.Items.Add("Измененный массив")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
```

Приложение 12

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. В новый массив перенести четные элементы исходного массива. Исходный и сформированный массивы вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim b() As Integer
Dim k As Integer
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
                        "-й элемент массива"))
```

Оглавление

```
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
k = -1
For i = 0 To n
    If a(i) Mod 2 = 0 Then
        k += 1
        ReDim Preserve b(k)
        b(k) = a(i)
    End If
Next
lstA.Items.Add("------------------------------")
If k = -1 Then
    lstA.Items.Add("Новый массив пуст")
Else
    lstA.Items.Add("Новый массив")
    For i = 0 To k
        lstA.Items.Add(Str(i) + vbTab + Str(b(i)))
    Next
End If
```

Приложение 13

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Проверить, равны ли все элементы массива друг другу. Исходный массив и полученные результаты вывести в окно списка.

Эта задача решается двумя разными способами.

Способ 1.

```
Dim a() As Integer
Dim n, i As Integer
Dim test As Boolean
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество элементов"))
```

Оглавление

```
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
                        "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
test = True
For i = 1 To n
    If a(i) <> a(0) Then
        test = False
        Exit For
    End If
Next
lstA.Items.Add("------------------------------")
If test Then
   lstA.Items.Add("Все элементы одинаковые")
Else
   lstA.Items.Add("Есть несовпадающие элементы")
End If
```

Способ 2.

```
Dim a() As Integer
Dim n, i As Integer
Dim kol As Integer
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
```

Оглавление

```
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
                        "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
kol = 0
For i = 0 To n
    If a(i) = a(0) Then
         kol += 1
    End If
Next
If kol = n + 1 Then
   lstA.Items.Add("Все элементы одинаковые")
Else
   lstA.Items.Add("Есть несовпадающие элементы")
End If
```

Приложение 14

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Проверить, упорядочены ли элементы массива по возрастанию. Исходный массив и полученные результаты вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim test As Boolean
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
```

Оглавление

```
                    "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
test = True
For i = 0 To n - 1
    If a(i) > a(i + 1) Then
        test = False
        Exit For
    End If
Next
lstA.Items.Add("------------------------------")
If test Then
    lstA.Items.Add("Элементы массива упорядочены")
Else
    lstA.Items.Add("Элементы массива неупорядочены")
End If
```

Приложение 15

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Отсортировать элементы массива по возрастанию методом пузырька. Массив до и после преобразования вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim sort As Boolean
Dim z As Integer
lstA.Items.Clear()
Do
    n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
```

Оглавление

```
    a(i) = Val(InputBox("Введите " + Str(i) + _
                        "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
Do
    sort = True
    For i = 0 To n - 1
        If a(i) > a(i + 1) Then
            z = a(i)
            a(i) = a(i + 1)
            a(i + 1) = z
            sort = False
        End If
    Next
Loop Until sort
lstA.Items.Add("------------------------------")
lstA.Items.Add("Массив после сортировки")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
```

Приложение 16

Дан целочисленный массив. Количество элементов и их значения вводятся с клавиатуры. Упорядочить элементы массива по возрастанию методом линейной сортировки (методом поиска минимума). Массив до и после преобразования вывести в окно списка.

```
Dim a() As Integer
Dim n, i As Integer
Dim start As Integer
Dim imin, min As Integer
Dim z As Integer
lstA.Items.Clear()
```

Оглавление

```
Do
    n = Val(InputBox("Введите количество элементов"))
Loop Until n > 0
n -= 1
ReDim a(n)
For i = 0 To n
    a(i) = Val(InputBox("Введите " + Str(i) + _
                        "-й элемент массива"))
Next
lstA.Items.Add("Исходный массив")
lstA.Items.Add("Номер" + vbTab + "Значение")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
For start = 0 To n - 1
    min = a(start)
    imin = start
    For i = start + 1 To n
        If a(i) < min Then
            min = a(i)
            imin = i
        End If
    Next
    a(imin) = a(start)
    a(start) = min
Next
lstA.Items.Add("-------------------------------")
lstA.Items.Add("Массив после сортировки")
For i = 0 To n
    lstA.Items.Add(Str(i) + vbTab + Str(a(i)))
Next
```

Список литературы

1. Волчёнков Н.Г. Программирование на Visual Basic 6: В 3-х ч. Часть 1. – М.: ИНФРА-М, 2000. – 286 с.

Оглавление

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