Оглавление

1. Основные понятия

Строка – это некоторый набор символов, в том числе и пустой. Для обозначения строки используются кавычки ("). Пустая строка обозначается парой кавычек, между которыми нет ни одного символа, в том числе и пробела (`s =""`).

Строки описываются с помощью типа `String`. Каждая строка может содержать до двух миллиардов символов в формате Unicode. Объем памяти, занимаемой строковой переменной, зависит от количества символов в этой строке. Чем больше символов, тем больше памяти требуется для хранения этой строки.

Число символов в строке называется длиной строки. Длина пустой строки равна нулю. Все символы в строке последовательно пронумерованы. Нумерация идет слева направо и начинается с единицы. Номер символа в строке также называют позицией символа.

Подстрока – это любой фрагмент строки, состоящий хотя бы из одного символа исходной строки. Например, в строке "электростанция" содержится подстрока "рост". Левой подстрокой называется такая подстрока, которая начинается с первого символа исходной строки. В нашем случае левой подстрокой будет "элек". Правая подстрока – это подстрока, которая заканчивается последним символом исходной строки. В нашем примере это будет подстрока "станция".

Конкатенацией двух строк `s1` и `s2` называется строка `s`, для которой `s1` является левой подстрокой, `s2` – правой подстрокой, а длина строки `s` равна сумме длин строк `s1` и `s2`. Часто конкатенацию называют сложением или склейкой строк. В Visual Basic 2005 она обозначается знаком плюс.

```
s = s1 + s2
```
[1]

Строковое выражение – это либо отдельная строка, либо строка и строковое выражение, между которыми стоит знак конкатенации.

2. Основные функции обработки строк

- `Strings.Len(Строка)` – возвращает количество символов в строке, то есть ее длину. Например, `Strings.Len("Окно")` вернет значение 4.

[1] Также допускается следующая запись конкатенации.
```
s = s1 & s2
```

Оглавление

- `Strings.Left(Строка, Длина)` – выделяет левую подстроку указанной `Длины` из заданной `Строки`. Например, `Strings.Left("Пароход", 3)` вернет строку `"Пар"`.
- `Strings.Right(Строка, Длина)` – выделяет правую подстроку указанной `Длины` из заданной `Строки`. Например, `Strings.Right("Пароход", 3)` вернет строку `"ход"`.
- `Strings.Mid(Строка, Позиция, Длина)` – выделяет подстроку заданной `Длины` из исходной `Строки`, начиная с указанной `Позиции`. Например, `Strings.Mid("Пароход", 2, 4)` вернет строку `"арох"`. Параметр `Длина` может отсутствовать. В этом случае функция `Strings.Mid` возвращает правую подстроку, которая начинается с заданной `Позиции`. Например, `Strings.Mid("Пароход", 4)` вернет строку `"оход"`. Функция `Mid` может стоять не только справа от знака присваивания, но и слева. В этом случае префикс `Strings` не ставится. Если функция `Mid` стоит слева от знака присваивания, то соответствующая подстрока исходной Строки будет заменена на значение строкового выражения, стоящего в правой части оператора присваивания. При этом длина исходной строки не меняется. Примеры:

  1. ```
     s = "Иванов"
     Mid(s, 1, 4) = "Петр"
     ```
     После выполнения оператора присваивания в переменную `s` будет записано `"Петров"`. То есть первые четыре символа исходной строки будут заменены на четыре символа из другой строки.
  2. ```
     s = "Иванов"
     Mid(s, 1, 4) = "Пят"
     ```
     После выполнения оператора присваивания в переменную `s` будет записано `"Пятнов"`. Так как во второй строке всего три символа, то в исходной строке будут изменены только первые три символа, а четвертый останется неизменным.
  3. ```
     s = "Иванов"
     Mid(s, 1, 3) = "Александр"
     ```
     После выполнения оператора присваивания в переменную `s` будет записано `"Аленов"`. Первые три символа исходной строки меняются на

Оглавление
     ```

первые три символа второй строки. Остальная часть второй строки в преобразовании не участвует.

Другой пример использования функции `Strings.Mid` для посимвольной обработки строки рассмотрен в разделе 4.

- `Strings.InStr(Старт, Строка, Подстрока)` – ищет `Подстроку` в `Строке`, начиная с указанной стартовой позиции. Стартовую позицию можно не указывать, тогда процесс поиска начинается с первого символа строки. При этом функция записывается в следующем виде: `Strings.InStr(Строка, Подстрока)`. Не зависимо от формы записи результатом функции является число, которое показывает, с какой позиции начинается искомая подстрока. Символы строки нумеруются, начиная с единицы. Если искомая `Подстрока` не найдена, то функция возвращает ноль. Рассмотрим примеры.
  1. `Strings.InStr("абракадабра", "бра")` даст результат 2, так как первый раз подстрока "бра" начинается со второго символа строки.
  2. `Strings.InStr(4, "абракадабра", "бра")` даст результат 9, так как поиск начнется с четвертого символа строки и будет найден второй слог "бра", который начинается с девятого символа строки.
  3. `Strings.InStr("абракадабра", "кот")` даст результат 0, так как подстрока "кот" в слове "абракадабра" не встречается.
- `Strings.InStrRev(Строка, Подстрока, Старт)` тоже ищет `Подстроку` в `Строке`, начиная с указанной стартовой позиции, но поиск идет справа налево, от конца строки к началу. Стартовая позиция отсчитывается от левого края строки. Результатом работы функции будет номер позиции, с которой начинается искомая подстрока. Этот номер тоже отсчитывается от левого края строки. Рассмотрим примеры.
  1. `Strings.InStrRev("абракадабра", "бра")` даст результат 9, так как последнее вхождение подстроки "бра" начинается с девятого символа строки.
  2. `Strings.InStrRev("абракадабра", "бра", 7)` даст результат 2, так как поиск начнется с седьмого символа строки и пойдет к началу строки. Поэтому будет найден слог "бра", стоящий в начале строки.
- `Strings.StrReverse(Строка)` – записывает все символы строки в обратном порядке. Например, `Strings.StrReverse("книга")` вернет `"агинк"`.

Оглавление

- `Strings.Replace(Строка, Найти, Заменить, Старт, Количество, Метод сравнения)` – в заданной `Строке` функция ищет подстроку `Найти` и заменяет ее на подстроку `Заменить`. Поиск начинается с указанной стартовой позиции и выполняется не более чем требуемое `Количество` замен. Если стартовая позиция не указана, то поиск начинается с первого символа исходной строки. Если же стартовая позиция указана и отличается от единицы, то предшествующая ей часть строки будет удалена. Если не указано требуемое `Количество` замен, то будут выполнены все возможные замены. `Метод сравнения` определяет режим сравнения исходной `Строки` и подстроки `Найти`. Этот параметр может иметь одно из двух значений: `CompareMethod.Bynary` и `CompareMNethod.Text`. В первом случае строки сравниваются с учетом регистра, когда строчная и прописная буквы считаются разными. Этот режим выбирается по умолчанию. То есть если при вызове функции `Метод сравнения` не указан, то сравнение будет происходить с учетом регистра. Во втором случае строки сравниваются без учета регистра, при этом Visual Basic 2005 не делает различия между строчными и прописными буквами. Примеры.
    1. `Strings.Replace("Иванов Иван Иванович", "Иван", "Петр")` даст результат `"Петров Петр Петрович"`.
    2. `Strings.Replace("Иванов Иван Иванович", "Иван", "Петр", 1, 2)` даст результат `"Петров Петр Иванович"`.
    3. `Strings.Replace("Иванов Иван Иванович", "Иван", "Петр", 8, 1)` даст результат `"Петр Иванович"`.
- `Strings.UCase(Строка)` – преобразует все строчные буквы исходной `Строки` в прописные.
- `Strings.LCase(Строка)` – преобразует все прописные буквы исходной `Строки` в строчные. Примеры.
    1. `Strings.UCase("Visual Basic")` даст результат `"VISUAL BASIC"`.
    2. `Strings.LCase("Visual Basic")` даст результат `"visual basic"`.
- `Strings.StrConv(Строка, Режим преобразования)` – преобразует строчные и прописные буквы в исходной `Строке`. При этом может использоваться один из трех основных `Режимов преобразования`.

Оглавление

1. `vbStrConv.LowerCase` – все буквы преобразуются в строчные (аналог функции `Strings.LCase`).
2. `vbStrConv.UpperCase` – все буквы преобразуются в прописные (аналог функции `Strings.UCase`).
3. `vbStrConv.ProperCase` – первая буква каждого слова будет прописной, а остальные – строчными.

- `Strings.LTrim(Строка)` – удаляет все пробелы, стоящие в начале `Строки`, до первого символа, который не является пробелом.
- `Strings.RTrim(Строка)` – удаляет все пробелы, стоящие в конце `Строки`, до ближайшего символа, который не является пробелом.
- `Strings.Trim(Строка)` – удаляет все пробелы, стоящие в начале и в конце `Строки`. Пробелы, стоящие в середине `Строки`, остаются без изменений.
- `Strings.Space(Количество)` – формирует строку, состоящую из заданного `Количества` пробелов.
- `Strings.StrDup(Количество, Символ)` – формирует строку, состоящую из заданного `Количества` повторений указанного `Символа`.
- `Strings.LSet(Строка, Длина)` – формирует строку указанной `Длины`. Если исходная `Строка` длиннее, чем требуется, то ее правая часть теряется. Если исходная `Строка` короче, чем требуется, то она будет дополнена справа необходимым количеством пробелов. Другими словами, происходит выравнивание строки по левой границе.
- `Strings.RSet(Строка, Длина)` – формирует строку указанной `Длины`. Если исходная `Строка` длиннее, чем требуется, то ее правая часть теряется. Если исходная `Строка` короче, чем требуется, то она будет дополнена слева необходимым количеством пробелов. Другими словами, происходит выравнивание строки по правой границе.
- `Strings.StrComp(Строка1, Строка2, Режим сравнения)` – сравнивает две строки и возвращает один из трех результатов: 1 (если `Строка1` больше, чем `Строка2`), 0 (если строки равны) или -1 (если `Строка1` меньше, чем `Строка2`). `Режим сравнения` определяет способ сравнения строк. Этот параметр может иметь одно из двух значений: `CompareMethod.Bynary` и `CompareMNethod.Text`. В первом случае строки сравниваются с учетом регистра, когда строчная и прописная буквы считаются разными. Этот режим

Оглавление

выбирается по умолчанию. Во втором случае строки сравниваются без учета регистра, при этом Visual Basic 2005 не делает различия между строчными и прописными буквами. Правила сравнения двух строк будут подробно рассмотрены в разделе 8.

- `Val(Строка)` – функция пытается преобразовать указанную строку в число. Преобразование прерывается при первой же ошибке. Если исходная `Строка` начинается не с цифры, а с символа, то функция возвращает значение 0. Примеры различных преобразований приведены в таблице 1.

Таблица 1

Исходная строка	Результат функции `Val`
"12"	12
"12a"	12
"1a2"	1
"a12"	0
"аб"	0
"0"	0

- `Str(Число)` – преобразует число в строку. Перед положительными числами функция ставит один пробел.
- `Strings.Asc(Строка)` – возвращает ASCII[2] код первого символа `Строки`.
- `Strings.Chr(Код)` – возвращает символ, соответствующий заданному ASCII `коду`.
- `Strings.Split(Строка, Разделитель, Количество)` – формирует массив из слов, содержащихся в `Строке`. Каждое слово записывается в отдельный элемент массива. Признаком конца очередного слова является появление в строке подстроки `Разделитель`. Если `Разделитель` не указан, то признаком конца слова является пробел. Результирующий массив следует описывать как массив с неизвестной верхней границей. Нумерация слов в массиве идет с нуля. Если параметр `Количество` не указан, то каждое слово будет записано в отдельный элемент массива. Если же `Количество` задано, то результирующий массив будет состоять не более чем из требуемого `Количества` элементов. Если в строке

[2] ASCII (American Standard Code for Information Interchange) – американский стандартный код для обмена информацией. ASCII представляет собой кодировку для представления десятичных цифр, латинского и национального (в нашей стране – русского) алфавитов, знаков препинания и управляющих символов.

Оглавление

содержится больше слов, чем указано в параметре Количество, то в последнем элементе массива в виде одного «слова» будет храниться вся необработанная часть строки. Если в строке идут два разделителя подряд, например, два пробела, то в массив слов вставляется пустое слово, и соответствующий элемент массива будет содержать пустую строку. Пример использования функции Strings.Split рассмотрен в разделе 4.

- Strings.Join(Массив слов, Разделитель) – формирует строку из Массива слов, разделяя слова с помощью указанной строки – Разделителя. Если Разделитель не указан, то слова разделяются одним пробелом. За последним словом массива Разделитель не ставится. Пример использования функции Strings.Join рассмотрен в разделе 5.

3. Посимвольная обработка строки

Существует несколько способов обрабатывать строку символов. Один из них – посимвольная обработка. В этом случае строка рассматривается как массив символов. Соответственно, для ее обработки организуется цикл по всем символам строки. Так как нумерация символов начинается с единицы, то начальное значение счетчика берут равным единице, а не нулю, как при работе с одномерным массивом. Номер последнего символа совпадает с длиной строки, поэтому конечное значение счетчика записывают в виде Len(s), где s – обрабатываемая строка. В теле цикла строку обрабатывают так же, как и одномерный массив. С помощью функции Strings.Mid выделяют один символ, его анализируют и обрабатывают. В качестве примера посимвольной обработки рассмотрим следующую задачу. В строке заменить каждый символ, стоящий после буквы «а» на символ звездочка (*).

Для решения задачи нам потребуются две переменные. В строковой переменной s мы будем хранить обрабатываемую строку.

```
Dim s As String
```

Так как посимвольная обработка выполняется в цикле, то для его организации нам потребуется целочисленный счетчик i.

```
Dim i As Integer
```

Очищаем окно списка от предыдущих результатов работы программы.

```
lstText.Items.Clear()
```

Оглавление

Вводим строку, которую надо обработать. Обратите внимание, при вводе строки не используется преобразование `Val`. Более того, использование данного преобразования является логической ошибкой, не позволяющей получить правильные результаты.

```
s = InputBox("Введите строку")
```

В окно списка выводим исходную строку текста, чтобы пользователь мог контролировать результаты работы программы.

```
lstText.Items.Add(s)
```

Организуем цикл для посимвольной обработки строки. Обработку будем начинать с первого символа (так как нулевого символа не существует). Заканчивать обработку будем на предпоследнем символе строки, поскольку нам надо изменить символ, следующий за обрабатываемым. Если буква «а» является последним символом строки, то символ, стоящий за ней, обработать нельзя, так как его нет.

```
For i = 1 To Len(s) – 1
```

С помощью функции `Strings.Mid` выделяем один символ, стоящий на `i`-ой позиции. Анализируем этот символ.

```
    If Strings.Mid(s, i, 1) = "a" Or _
       Strings. Mid(s, i, 1) = "A" Then
```

Если этот символ является строчной или прописной буквой «а», то символ, стоящий за ним, необходимо заменить на звездочку. Так как мы анализировали символ, стоящий на `i`-ой позиции, то изменять мы будем символ, стоящий на позиции (`i+1`). Обратите внимание, что любой символ, явно используемый в программе, берется в кавычки.

```
            Mid(s, i + 1, 1) = "*"
    End If
Next
```

После завершения основного цикла нам остается только вывести измененную строку в окно списка.

```
lstText.Items.Add(s)
```

Пример работы программы приведен на рис. 1.

Оглавление

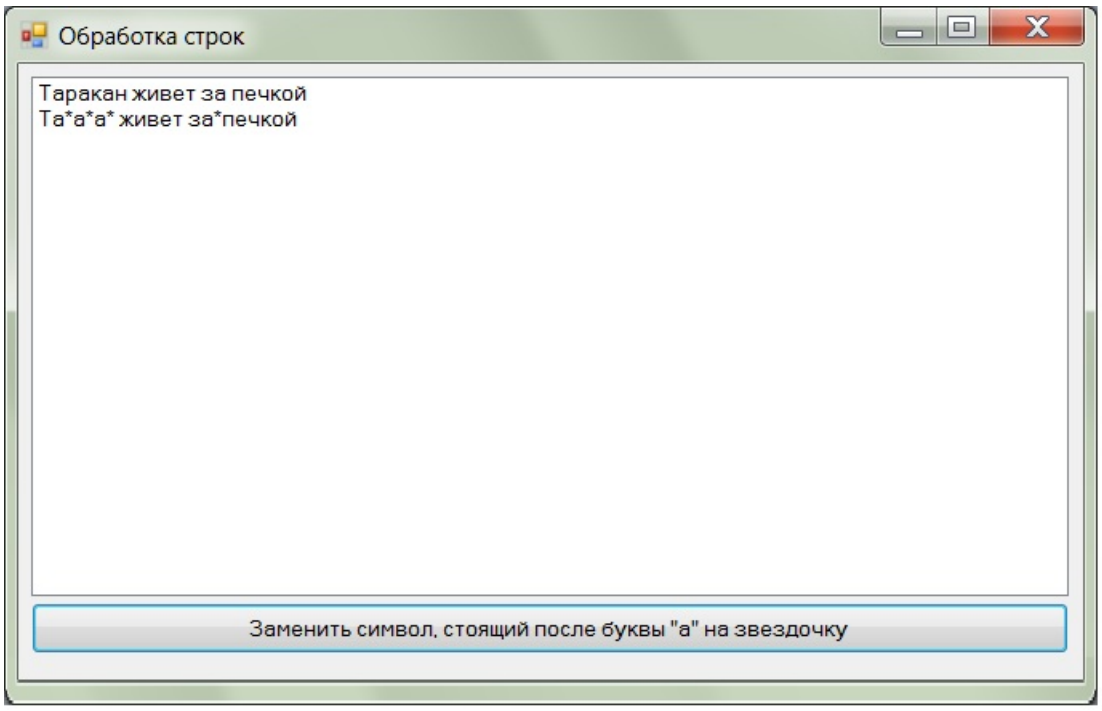

Рис. 1. Пример посимвольной обработки строки

4. Формирование массива слов строки

Формирование массива слов строки является одной из базовых операций при обработке символьной информации. Решение большинства строковых задач начинается с ввода строки и ее разбиения на массив слов. Такой подход позволяет существенно упростить решение задач, в которых требуется найти слово, удовлетворяющее какому-либо условию. В общем случае задача формирования массива слов формулируется следующим образом.

Дана строка текста, в которой слова разделены произвольным количеством пробелов. Пробелы могут стоять перед первым словом и за последним. Сформировать массив слов строки. Исходную строку и полученный массив вывести в окно списка.

Существует три различных способа решения этой задачи. Рассмотрим особенности программной реализации каждого из них.

Способ 1.

Первый способ основан на посимвольной обработке строки. Предполагается, что внутри слова нет пробелов. Поэтому каждый символ строки анализируется. Если он не является пробелом, значит, это очередная буква слова. Тогда она дописывается в конец слова, выделяемого в данный момент. Если очередной символ не является пробелом, а сразу за ним стоит пробел, значит, это последний символ слова.

Оглавление

Полученное слово надо дописать в массив слов, а текущее слово сделать пустым, так как оно еще не началось.

Для решения задачи нам потребуются следующие переменные. В строковой переменной `s` мы будем хранить обрабатываемую строку.

```
Dim s As String
```

Так как посимвольная обработка выполняется в цикле, то для его организации нам потребуется целочисленный счетчик `i`.

```
Dim i As Integer
```

Выделяемые слова мы будем записывать в одномерный массив `slova()`. Этот массив обязательно будет иметь тип `String`.

```
Dim slova() As String
```

Так как каждое слово мы будем выделять посимвольно, то нам потребуется строковая переменная, в которой будет постепенно накапливаться очередное слово строки.

```
Dim tek As String
```

Для обработки массива необходимо знать, сколько элементов в нем содержится. Номер последнего элемента, записанного в массив слов, будем хранить в переменной `n`.

```
Dim n As Integer
```

Очищаем окно списка от предыдущих результатов работы программы.

```
lstText.Items.Clear()
```

Вводим строку, которую надо обработать. Обратите внимание, при вводе строки не используется преобразование `Val`. Более того, использование данного преобразования является логической ошибкой, не позволяющей получить правильные результаты.

```
s = InputBox("Введите строку")
```

В окно списка выводим исходную строку текста, чтобы пользователь мог контролировать результаты работы программы.

```
lstText.Items.Add(s)
```

До начала формирования массив слов пуст. В нем нет ни одного элемента. Поэтому номер последнего элемента находится за пределами массива. Возьмем его равным -1.

```
n = -1
```

Перед началом выделения слов из строки очистим переменную, в которой будет находиться очередное слово.

```
tek = ""
```

Алгоритм предполагает, что за каждым словом строки обязательно идет пробел. Чтобы последнее слово тоже подчинялось этому правилу, допишем к исходной строке пробел.

Оглавление

```
s += " "
```

Организуем цикл для посимвольной обработки строки. Обработку будем начинать с первого символа (так как нулевого символа не существует). Заканчивать обработку будем на предпоследнем символе строки, поскольку мы анализируем не только текущий символ строки, но и следующим за ним. А для последнего символа следующий символ не существует.

```
For i = 1 To Len(s) – 1
```

С помощью функции `Strings.Mid` выделяем один символ, стоящий на `i`-ой позиции. Анализируем этот символ.

```
If Mid(s, i, 1) <> " " Then
```

Если выделенный символ не является пробелом, то дописываем его к текущему слову.

```
tek += Mid(s, i, 1)
End If
```

Теперь анализируем текущий символ и символ, следующий за ним.

```
If Mid(s, i, 1) <> " " And _
   Mid(s, i + 1, 1) = " " Then
```

Если текущий символ не является проблем, а за ним следует пробел, значит, текущий символ был последним в слове. Следовательно, очередное слово строки выделено полностью, и его надо записать в массив слов. При этом номер последнего слова в массиве увеличится на единицу.

```
n += 1
```

Изменяем размер массива. Использование ключевого слова `Preserve` позволяет сохранить ранее найденные слова.

```
ReDim Preserve slova(n)
```

В массив слов на последнюю позицию записываем выделенное слово.

```
slova(n) = tek
```

Так как следующее слово еще не началось, полагаем его пустым.

```
tek = ""
End If
Next
```

После завершения цикла массив слов полностью сформирован, и нам остается только вывести его в окно списка. Сначала выводим горизонтальную черту, чтобы зрительно отделить исходную строку от полученного массива.

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

Оглавление

Затем выводим поясняющий заголовок.

```
lstText.Items.Add("Слова строки")
```

Теперь мы организуем цикл для вывода всех элементов массива слов. Обратите внимание, что элементы массива слов нумеруются с нуля.

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

На каждом шаге цикла выводим очередное слово в окно списка.

```
    lstText.Items.Add(slova(i))
Next
```

После завершения цикла вывода массив слов можно обрабатывать как любой одномерный массив.

Способ 2.

Второй способ предлагает обрабатывать строку с помощью функций Visual Basic 2005. Сначала из строки удаляются все пробелы, стоящие перед первым словом и за последним. Затем убираются лишние пробелы, чтобы между словами осталось ровно по одному пробелу. Наконец к строке дописывается равно один пробел, чтобы за каждым словом строки шел пробел, в том числе и за последним. После этого переходят к выделению слов. Ищут позицию первого пробела в строке. Все, что стоит до пробела, является первым словом. Его записывают в массив и удаляют из основной строки вместе с пробелом. Процесс повторяется многократно до тех пор, пока в строке не останется ни одного слова.

Для решения задачи нам потребуются следующие переменные. В строковой переменной `s` мы будем хранить обрабатываемую строку.

```
Dim s As String
```

Так как массив слов выводится в цикле, то для его организации нам потребуется целочисленный счетчик `i`.

```
Dim i As Integer
```

Выделяемые слова мы будем записывать в одномерный массив `slova()`. Этот массив обязательно будет иметь тип `String`.

```
Dim slova() As String
```

Для обработки массива необходимо знать, сколько элементов в нем содержится. Номер последнего элемента, записанного в массив слов, будем хранить в переменной `n`.

```
Dim n As Integer
```

Оглавление

В процессе выделения очередного слова мы будем искать в строке первый пробел. Его позицию мы будем хранить в целочисленной переменной `pos`.

```
Dim pos As Integer
```

Очищаем окно списка от предыдущих результатов работы программы.

```
lstText.Items.Clear()
```

Вводим строку, которую надо обработать. Обратите внимание, при вводе строки не используется преобразование `Val`. Более того, использование данного преобразования является логической ошибкой, не позволяющей получить правильные результаты.

```
s = InputBox("Введите строку")
```

В окно списка выводим исходную строку текста, чтобы пользователь мог контролировать результаты работы программы.

```
lstText.Items.Add(s)
```

Удаляем пробелы, стоящие перед первым словом и за последним.

```
s = Strings.Trim(s)
```

Удаляем лишние пробелы между словами. Для этого во всей строке многократно заменяем два подряд идущих пробела на один. Процесс замены повторяется до тех пор, пока в строке не останется ни одного сдвоенного пробела. Другими словами, цикл завершится тогда, когда функция `Strings.InStr`, ищущая два подряд идущих пробела, вернет значение ноль, которое означает, что в заданной строке нет искомой подстроки.

```
Do Until Strings.InStr(s, "  ") = 0
```

Функция `Strings.Replace` используется для замены всех сдвоенных пробелов одинарными.

```
        s = Strings.Replace(s, "  ", " ")
Loop
```

До начала формирования массив слов пуст. В нем нет ни одного элемента. Поэтому номер последнего элемента находится за пределами массива. Возьмем его равным -1.

```
n = -1
```

Чтобы последнее слово тоже подчинялось этому правилу, допишем к исходной строке пробел.

```
s += " "
```

Процесс выделения слов продолжается до тех пор, пока в исходной строке есть хотя бы одно слово. При этом ее длина обязательно будет больше нуля. Как только в строке не

Оглавление

останется ни одного слова, ее длина станет равной нулю, и выполнение цикла прекратится.

```
Do While Len(s) > 0
```

Ищем в строке первый пробел и запоминаем его позицию.

```
    pos = Strings.InStr(s, " ")
```

Переходим выделению очередного слова. Так как новое слово будет записано в массив, то номер последнего элемента в этом массиве увеличится на единицу.

```
    n += 1
```

Изменяем размер массива. Использование ключевого слова `Preserve` позволяет сохранить ранее найденные слова.

```
    ReDim Preserve slova(n)
```

На место последнего элемента массива записываем очередное слово. Это слово является левой подстрокой обрабатываемой строки, которая закончится перед пробелом.

```
    slova(n) = Strings.Left(s, pos - 1)
```

Теперь удаляем выделенное слово из строки. Для этого достаточно выделить правую подстроку обрабатываемой строки, начиная с символа, стоящего сразу после пробела.

```
    s = Strings.Mid(s, pos + 1)
Loop
```

После завершения цикла массив слов полностью сформирован, и нам остается только вывести его в окно списка. Сначала выводим горизонтальную черту, чтобы зрительно отделить исходную строку от полученного массива.

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

Затем выводим поясняющий заголовок.

```
lstText.Items.Add("Слова строки")
```

Теперь мы организуем цикл для вывода всех элементов массива слов. Обратите внимание, что элементы массива слов нумеруются с нуля.

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

На каждом шаге цикла выводим очередное слово в окно списка.

```
    lstText.Items.Add(slova(i))
Next
```

После завершения цикла вывода массив слов можно обрабатывать как любой одномерный массив.

Оглавление

Способ 3.

Третий способ формирования массива слов основан на использовании функции `Strings.Split`, описанной в разделе 2. Эта функция разбивает строку на слова, которые записывает в массив. Перед использованием этой функции строку необходимо преобразовать: удалить из нее все пробелы, стоящие перед первым словом и за последним, и убрать лишние пробелы между словами. То есть слова в строке должны быть разделены ровно одним пробелом.

Для решения задачи нам потребуются следующие переменные. В строковой переменной `s` мы будем хранить обрабатываемую строку.

```
Dim s As String
```

Так как массив слов выводится в цикле, то для его организации нам потребуется целочисленный счетчик `i`.

```
Dim i As Integer
```

Выделяемые слова мы будем записывать в одномерный массив `slova()`. Этот массив обязательно будет иметь тип `String`.

```
Dim slova() As String
```

Для обработки массива необходимо знать, сколько элементов в нем содержится. Номер последнего элемента, записанного в массив слов, будем хранить в переменной `n`.

```
Dim n As Integer
```

Очищаем окно списка от предыдущих результатов работы программы.

```
lstText.Items.Clear()
```

Вводим строку, которую надо обработать. Обратите внимание, при вводе строки не используется преобразование `Val`. Более того, использование данного преобразования является логической ошибкой, не позволяющей получить правильные результаты.

```
s = InputBox("Введите строку")
```

В окно списка выводим исходную строку текста, чтобы пользователь мог контролировать результаты работы программы.

```
lstText.Items.Add(s)
```

Удаляем пробелы, стоящие перед первым словом и за последним.

```
s = Strings.Trim(s)
```

Удаляем лишние пробелы между словами. Для этого во всей строке многократно заменяем два подряд идущих пробела на один. Процесс замены повторяется до тех пор, пока в строке не останется ни одного сдвоенного пробела. Другими словами, цикл завершится тогда, когда функция `Strings.InStr`, ищущая два подряд идущих

Оглавление

пробела, вернет значение ноль, которое означает, что в заданной строке нет искомой подстроки.

```
Do Until Strings.InStr(s, "  ") = 0
```

Функция `Strings.Replace` используется для замены всех сдвоенных пробелов одинарными.

```
        s = Strings.Replace(s, "  ", " ")
Loop
```

Теперь можно вызвать функцию `Strings.Split`, которая сформирует массив слов. Так как в нашей строке слова разделяются пробелом, то его в качестве разделителя можно не указывать.

```
slova = Strings.Split(s)
```

Чтобы иметь возможность обрабатывать массив слов нам необходимо узнать номер последнего элемента в этом массиве. Для этого в Visual Basic 2005 есть специальная функция `UBound`. Она возвращает номер последнего элемента массива, который указан в круглых скобках и, следовательно, является единственным аргументом этой функции.

```
n = UBound(slova)
```

Теперь массив слов полностью сформирован, и нам остается только вывести его в окно списка. Сначала выводим горизонтальную черту, чтобы зрительно отделить исходную строку от полученного массива.

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

Затем выводим поясняющий заголовок.

```
lstText.Items.Add("Слова строки")
```

Теперь мы организуем цикл для вывода всех элементов массива слов. Обратите внимание, что элементы массива слов нумеруются с нуля.

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

На каждом шаге цикла выводим очередное слово в окно списка.

```
        lstText.Items.Add(slova(i))
Next
```

После завершения цикла вывода массив слов можно обрабатывать как любой одномерный массив.

Пример работы программы приведен на рис. 2.

Оглавление

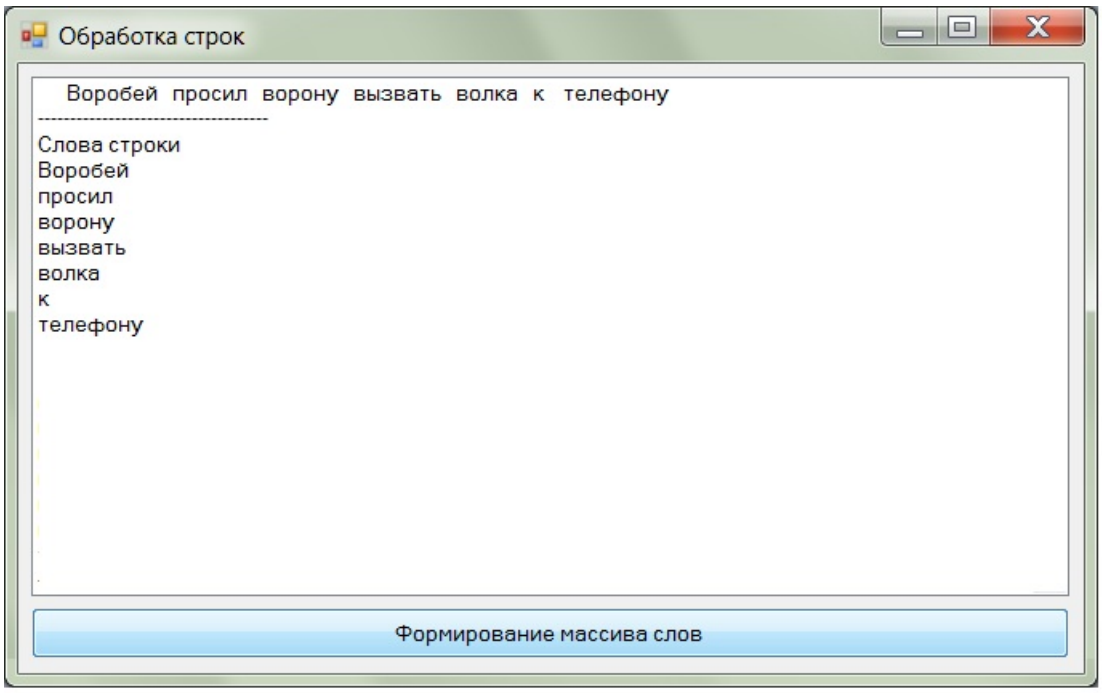

Рис. 2. Пример работы программы формирования массива слов

5. Формирование строки из массива слов

Из массива слов всегда можно сформировать строку, которая будет отличаться от исходной строки только количеством пробелов между словами. Эта задача имеет два различных решения.

Способ 1.

Строка формируется по алгоритму вычисления суммы всех элементов одномерного массива. Перед началом цикла результирующая строковая переменная очищается, в нее записывается пустая строка. На каждом шаге цикла к строке добавляется очередной элемент массива слов и один пробел. При этом пробел обязательно берется в кавычки. После завершения цикла в строковой переменной хранится уже сформированная строка, за последним словом которой стоит один пробел. Его можно удалит с помощью функции `Strings.RTrim`. Рассмотрим особенности программной реализации этого алгоритма.

Объявляем переменную, в которой будет формироваться строка.

```
Dim s as String
```

Для работы с массивом слов необходимо использовать цикл. Для его организации нам потребуется целочисленный счетчик `i`.

```
Dim i As Integer
```

Оглавление

Очищаем результирующую строку. До начала обработки массива слов в ней ничего нет.

```
s = ""
```

Организуем цикл для обработки всех элементов массива слов. Обратите внимание, что элементы массива слов нумеруются с нуля.

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

На каждом шаге цикла к результирующей строке добавляем очередное слово и пробел. Пробел обязательно берется в кавычки.

```
    s += slova(i) + " "
Next
```

После завершения цикла строка уже полностью сформирована. Нам остается только удалить пробел, стоящий за последним словом.

```
s = Strings.RTrim(s)
```

Полученную строку выводим в окно списка.

```
lstText.Items.Add(s)
```

Способ 2.

Второй способ формирования строки из массива слов основан на использовании стандартной функции Visual Basic 2005 `Strings.Join`, которая описана в разделе 2. Эта функция формирует строку из массива слов, разделяя слова указанной подстрокой – разделителем. Разделитель можно и не указывать, тогда слова будут разделены ровно одним пробелом. За последним словом разделитель не ставится. Рассмотрим пример использования функции `Strings.Join`.

Объявляем переменную, в которой будет храниться результирующая строка.

```
Dim s as String
```

Вызываем функцию `Strings.Join` для формирования строки из массива слов. Так как разделитель не указан, то слова будет разделены одним пробелом.

```
s = Strings.Join(slova)
```

Полученную строку выводим в окно списка.

```
lstText.Items.Add(s)
```

Пример использования функции рассмотрен в разделе 8 и проиллюстрирован на рис. 5.

Оглавление

6. Слова-палиндромы

Палиндромом называется слово, которое одинаково читается как слева направо, так и наоборот. Рассмотрим фрагмент программы, которая в произвольной строке ищет самое длинное слово-палиндром.

В начале программы вводится строка текста. Затем из нее удаляются лишние пробелы, и формируется массив слов. Массив называется `slova()`, номер последнего элемента в этом массиве хранится в переменной `n`.

Для решения задачи нам потребуется дополнительная переменная `max`, в которой будет храниться самое длинное слово-палиндром. Очевидно, что эта переменная будет иметь строковый тип.

```
Dim max As String
```

В качестве начального значения максимума возьмем пустую строку, так как это самое короткое из всех слов.

```
max = ""
```

Организуем цикл по всем словам строки. Слова в массиве нумеруются с нуля.

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

Анализируем очередное слово.

```
    If slova(i) = Strings.StrReverse(slova(i)) _
       And Len(slova(i)) > Len(max) Then
```

С помощью функции `Strings.StrReverse` записываем буквы слов в обратном порядке. Если получившееся слово совпадает с исходным, значит, текущее слово является палиндромом. Если при этом длина слова больше длины ранее найденного максимума, значит, значение максимума надо обновить, записав в него анализируемое слово.

```
            max = slova(i)
    End If
Next
```

После завершения цикла в переменной `max` будет храниться самое длинное слово–палиндром. Нам остается только распечатать результаты. Сначала выведем горизонтальную черту, чтобы зрительно отделить результаты работы программы от исходных данных.

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

Теперь анализируем полученный результат.

```
If max <> "" Then
```

Оглавление

Если значение переменной `max` отличается от пустой строки, значит, в исходной строке были слова-палиндромы. Выводим поясняющий текст и найденное слово.

```
            lstText.Items.Add("Самое длинное слово-палиндром")
            lstText.Items.Add(max)
        Else
```

Иначе, если в переменной max записана пустая строка, мы делаем вывод, что в строке не было ни одного слова-палиндрома. Следовательно, нет и самого длинного палиндрома. В этом случае мы выводим поясняющее сообщение.

```
            lstText.Items.Add("В строке нет слов-палиндромов")
        End If
```

Полный текст программы представлен в приложении 1. Пример работы программы приведен на рис. 3.

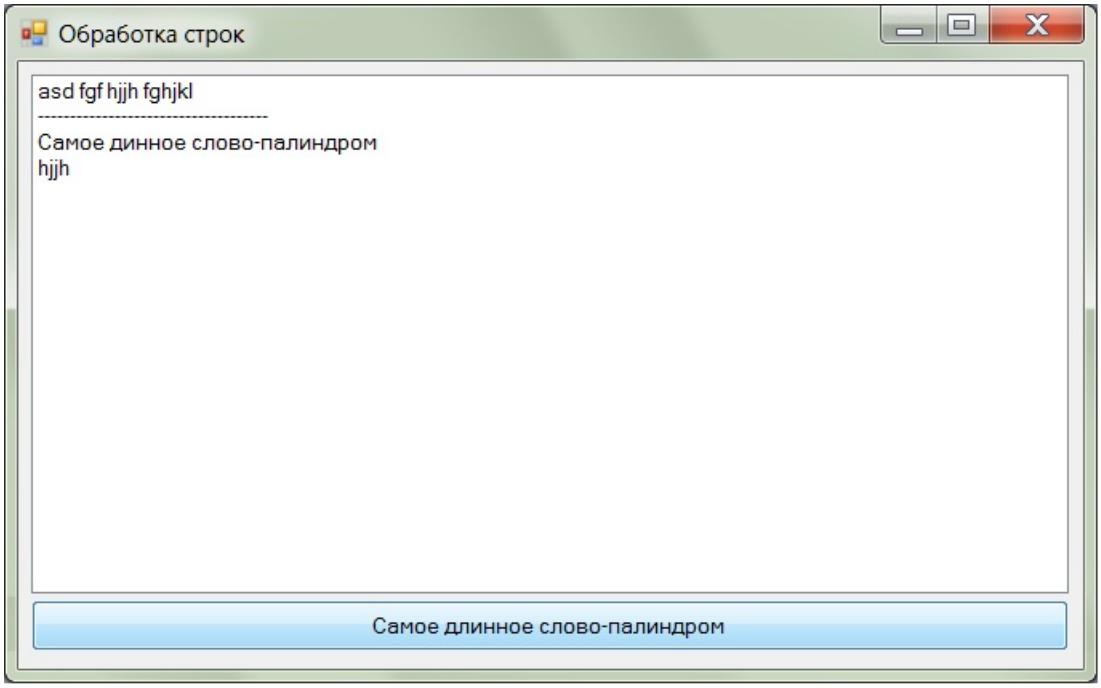

Рис. 3. Пример работы программы обработки слов-палиндромов

7. Выделение чисел из строки

При анализе слов строки часто возникает необходимость определить, является ли очередное слово числом или нет. Для этого используют следующую комбинацию функций `Val` и `Str`: `Str(Val(слово))`. Если слово, указанное в качестве аргумента функции `Val`, является числом, то оно не изменится в результате такого преобразования. Но если это число окажется положительным, то перед ним функция

Оглавление

`Str` обязательно добавит один пробел, который потом может быть удален функцией `Strings.Trim`. Все возможнее варианты преобразований приведены в таблице 2.

Таблица 2

`Slovo` Исходная строка	`Val (Slovo)`	`Str (Val(Slovo))`	`Strings.Trim (Str(Val(Slovo)))`
"12"	12	" 12"	"12"
"12a"	12	" 12"	"12"
"1a2"	1	" 1"	"1"
"a12"	0	" 0"	"0"
"аб"	0	" 0"	"0"
"0"	0	" 0"	"0"
"-45"	-45	"-45"	"-45"

Заметим, что исходная строка совпадает с преобразованной только в трех случаях: когда `Slovo` равно "12", "0" и "-45". Значит, если выполняется условие

```
Strings.Trim(Str(Val(slovo))) = slovo
```

то анализируемое слово является числом.

В качестве примера использования этого условия рассмотрим задачу вычисления суммы всех чисел, которые встречаются в произвольной строке текста.

В начале программы вводится строка текста. Затем из нее удаляются лишние пробелы, и формируется массив слов. Массив называется `slova()`, номер последнего элемента в этом массиве хранится в переменной `n`.

Для решения задачи нам потребуются две переменные. В переменной `summa` будет накапливаться сумма всех чисел, встретившихся в строке, а переменная `kol` нам нужна для подсчета количества этих чисел.

```
Dim summa, kol As Integer
```

Задаем начальные значения переменным. Так как до начала анализа массива слов нам не встретилось ни одного числа, то сумма и количество чисел задаются равными нулю.

```
summa = 0
kol = 0
```

Организуем цикл для обработки массива слов. Обратите внимание, что слова в массиве нумеруются с нуля.

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

Анализируем очередной элемент массива.

```
If Strings.Trim(Str(Val(slova(i)))) = _
```

Оглавление

```
            slova(i) Then
```

Если для него выполняется рассмотренное ранее условие, значит, данное слово является числом, и его необходимо включить в общую сумму. Но перед этим слово необходимо преобразовать в число с помощью функции `Val`.

```
                summa += Val(slova(i))
```

При этом количество найденных чисел увеличивается на единицу.

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

После завершения цикла надо распечатать результаты. Сначала выведем горизонтальную черту, чтобы зрительно отделить результаты работы программы от исходных данных.

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

Анализируем количество найденных чисел.

```
        If kol = 0 Then
```

Если количество чисел равно нулю, значит, в строке не было ни одного числа, и вычислить их сумму невозможно. В этом случае надо вывести поясняющее сообщение.

```
            lstText.Items.Add("В строке нет чисел")
        Else
```

Иначе печатаем найденную сумму чисел.

```
            lstText.Items.Add("Сумма =" + Str(summa))
        End If
```

Полный текст программы представлен в приложении 2. Пример работы программы приведен на рис. 4.

Оглавление

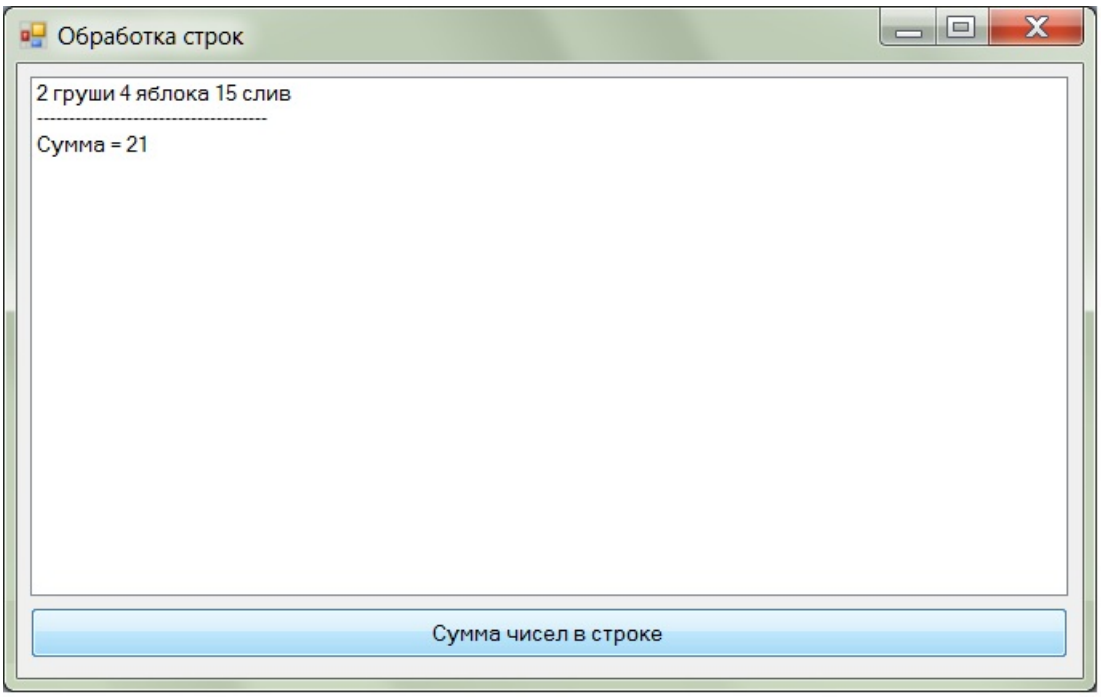

Рис. 4. Пример работы программы выделения чисел из строки

8. Сравнение строк

Сравнение строк в Visual Basic 2005 выполняется посимвольно. Сначала сравниваются первые символы строк. Если первые символы равны, сравниваются вторые, замет третьи и так далее. Вывод о том, какая строка больше, делается, как только будет найдена первая пара несовпадающих символов. Символы сравниваются по ASCII кодам. Меньшим считается символ, у которого меньше ASCII код. На основании таблицы ASCII кодов можно сформулировать несколько правил сравнения строк.

1. Пустая строка является минимальной возможной строкой. Любая непустая строка больше пустой.
2. Пробел меньше всех остальных символов.
3. Цифры меньше любой буквы.
4. Цифры идут по порядку от 0 до 9.
5. Любая латинская буква меньше любой русской.
6. В пределах каждого алфавита (латинского или русского) прописная буква всегда меньше строчной.
7. В пределах каждого алфавита буквы располагаются по порядку: в латинском алфавите – от A до Z, в русском алфавите от А до Я.

Оглавление

В таблице 3 приведены основные группы символов по возрастанию их ASCII кодов.

Таблица 3

Номер группы	Описание группы	Символы
1	Пробел	" "
2	Цифры	"0" – "9"
3	Латинские прописные буквы	"A" – "Z"
4	Латинские строчные буквы	"a" – "z"
5	Русские прописные буквы	"А" – "Я"
6	Русские строчные буквы	"а" – "я"

Пользуясь данными таблицы 3, сравним несколько строк. Результаты сравнения представлены в таблице 4.

Таблица 4

Результат сравнения	Пояснение
"Арбуз" < "Банан"	Так как буква "А" стоит в алфавите перед буквой "Б".
"Белка" > "Бегемот"	Первые и вторые буквы слов совпадают, сравнение происходит по третьим буквам. Буква "г" стоит в алфавите перед буквой "л", следовательно, слово "Бегемот" меньше, чем слово "Белка".
"Пар" < "Пароход"	Первое слово является левой подстрокой второго. Когда из второго слова для сравнения берется четвертая буква, из первого слова берется пустая строка, которая меньше любой другой строки. Следовательно, левая подстрока всегда будет меньше всей строки.
"Лошадь" < "воробей"	Первое слово начинается с прописной буквы, а второе – со строчной. В пределах одного алфавита любая прописная буква меньше любой строчной. Следовательно, слово "Лошадь" меньше, чем слово "воробей".

В качестве примера приведем фрагмент программы, сортирующей слова в произвольной строке. Эту задачу можно решать в двух вариантах. При сортировке с

Оглавление

учетом регистра делается различие меду строчными и прописными буквами. При сортировке без учета регистра строчные и прописные буквы не различаются. Рассмотрим оба случая.

Сортировка с учетом регистра.

В начале программы вводится строка текста. Затем из нее удаляются лишние пробелы, и формируется массив слов. Массив называется `slova()`, номер последнего элемента в этом массиве хранится в переменной `n`.

Сортировать массив слов будем методом пузырька. Поэтому нам потребуются две дополнительные переменные. В переменной `sort` мы будем хранить состояние массива: отсортирован он или нет. Эта переменная будет иметь логический тип данных.

```
Dim sort As Boolean
```

Любая сортировка предполагает перестановку элементов массива местами. Для этого необходима дополнительная переменная. Ее тип обязательно должен совпадать с типом элементов сортируемого массива. Так как мы сортируем массив слов, то дополнительная переменная должна иметь строковый тип данных.

```
Dim z As String
```

Организуем внешний цикл сортировки массива.

```
Do
```

Сначала предполагаем, что массив уже отсортирован.

```
    sort = True
```

Анализируем все слова, хранящиеся в массиве. Обратите внимание, что слова в массиве нумеруются с нулю. Процесс анализа необходимо остановить после обработки предпоследнего слова, которое сравнивается с последним.

```
    For i = 0 To n – 1
```

Сравниваем текущее слово и следующее за ним. Так как проверка выполняется с учетом регистра букв, то никаких дополнительных условий ставить не надо.

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

Если `i`-е слово больше, чем (`i+1`)-е, значит, массив неупорядочен. Меняем значение логической переменной.

```
            sort = False
```

После чего переставляем местами элементы массива, используя дополнительную переменную.

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

Оглавление

```
                slova(i + 1) = z
            End If
        Next
```

Процесс перестановки слов в массиве продолжается до тех пор, пока все элементы массива не займут правильные места.

```
    Loop Until sort
```

Теперь из отсортированного массива слов формируем строку. В качестве разделителя слов используем пробел.

```
    s = Strings.Join(slova, " ")
```

Полученные результаты выводим в окно списка. Чтобы зрительно отделить исходные данные от полученных результатов сначала выводим горизонтальную черту.

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

Потом печатаем полученную строку, в которой все слова упорядочены с учетом регистра.

```
    lstText.Items.Add(s)
```

Полный текст программы представлен в приложении 3. Пример работы программы приведен на рис. 5. Обратите внимание на то, что слово «Цапля», на первый взгляд, нарушает алфавитный порядок. Это вызвано тем, что оно начитается с прописной буквы, которая всегда меньше любой строчной.

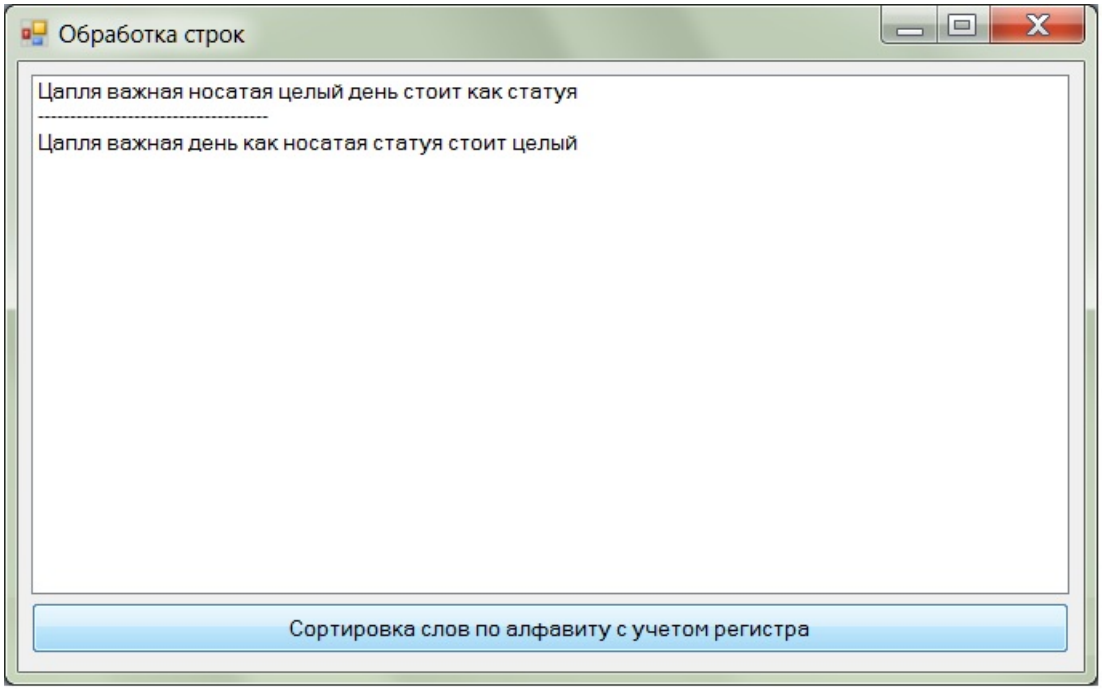

Рис. 5. Пример работы программы сортировки слов с учетом регистра

Оглавление

Сортировка без учета регистра.

В начале программы вводится строка текста. Затем из нее удаляются лишние пробелы, и формируется массив слов. Массив называется `slova()`, номер последнего элемента в этом массиве хранится в переменной `n`.

Сортировать массив слов будем методом пузырька. Поэтому нам потребуются две дополнительные переменные. В переменной `sort` мы будем хранить состояние массива: отсортирован он или нет. Эта переменная будет иметь логический тип данных.

```
Dim sort As Boolean
```

Любая сортировка предполагает перестановку элементов массива местами. Для этого необходима дополнительная переменная. Ее тип обязательно должен совпадать с типом элементов сортируемого массива. Так как мы сортируем массив слов, то дополнительная переменная должна иметь строковый тип данных.

```
Dim z As String
```

Организуем внешний цикл сортировки массива.

```
Do
```

Сначала предполагаем, что массив уже отсортирован.

```
    sort = True
```

Анализируем все слова, хранящиеся в массиве. Обратите внимание, что слова в массиве нумеруются с нулю. Процесс анализа необходимо остановить после обработки предпоследнего слова, которое сравнивается с последним.

```
    For i = 0 To n – 1
```

Сравниваем текущее слово и следующее за ним. Нам требуется сравнивать слова без учета регистра. Поэтому оба сравниваемых слова преобразуем таким образом, чтобы все их буквы были прописными. Для этого мы будем использовать функцию `Strings.UCase`.

```
        If Strings.UCase(slova(i)) > _
           Strings.UCase(slova(i + 1)) Then
```

Если `i`-е слово больше, чем (`i+1`)-е, значит, массив неупорядочен. Меняем значение логической переменной.

```
            sort = False
```

После чего переставляем местами элементы массива, используя дополнительную переменную.

```
            z = slova(i)
```

Оглавление

```
                slova(i) = slova(i + 1)
                slova(i + 1) = z
            End If
        Next
```

Процесс перестановки слов в массиве продолжается до тех пор, пока все элементы массива не займут правильные места.

```
    Loop Until sort
```

Теперь из отсортированного массива слов формируем строку. В качестве разделителя слов используем пробел.

```
    s = Strings.Join(slova, " ")
```

Полученные результаты выводим в окно списка. Чтобы зрительно отделить исходные данные от полученных результатов сначала выводим горизонтальную черту.

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

Потом печатаем полученную строку, в которой все слова упорядочены без учета регистра.

```
    lstText.Items.Add(s)
```

Полный текст программы представлен в приложении 3. Пример работы программы приведен на рис. 6. Обратите внимание на положения слова «Цапля» в результирующей строке. Сравните результаты, представленные на рисунках 5 и 6.

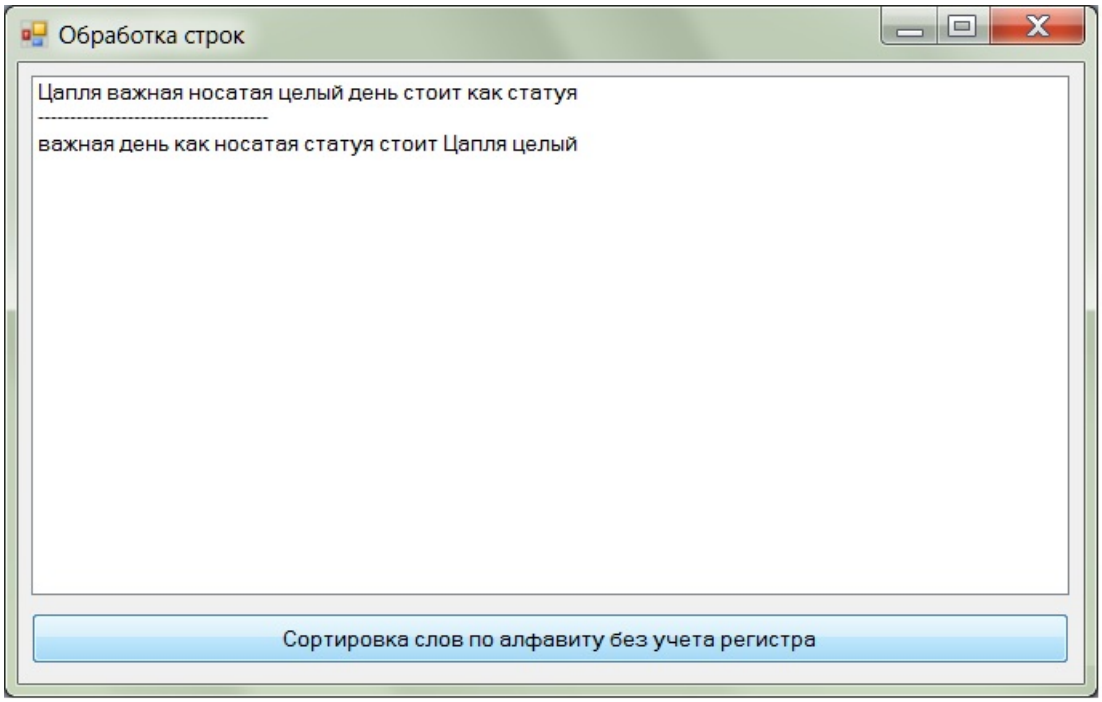

Рис. 6. Пример работы программы сортировки слов без учета регистра

Оглавление

9. Обработка многострочного текста

Многострочный текст – это одномерный массив строк. Каждый элемент массива – это отдельная строка. Следовательно, для обработки многострочного текста необходимо использовать цикл, который организуется таким образом, чтобы на каждом шаге цикла полностью обрабатывать одну строку исходного текста. В остальном обработка многострочного текста ничем не отличается от обработки отдельной строки.

В качестве примера рассмотрим следующую задачу.

Дан текст, состоящий из нескольких строк. В каждой строке слова разделены произвольным количеством пробелов. Пробелы могут стоять перед первым словом и за последним. В каждой строке текста надо записать слова в обратном порядке. Исходный и преобразованный тексты вывести в окно списка.

При решении задачи нам потребуется одномерный массив строк для хранения исходного текста. Так как количество строк заранее неизвестно, то опишем массив без указания номера последнего элемента.

```
Dim Tekst() As String
```

Для организации цикла нам необходимо знать номер последней строки текста. Он будет храниться в целочисленной переменной `n`.

```
Dim n As Integer
```

По условию задачи к каждой строке текста надо переставить слова. Поэтому для каждой строки исходного текста мы будем формировать массив слов. Следовательно, надо объявить одномерный строковый массив, в который мы будем записывать слова.

```
Dim slova() As String
```

Для обработки массива нам необходимо знать номер последнего слова, который мы будем хранить в специальной целочисленной переменной `k`.

```
Dim k As Integer
```

Для организации циклов нам потребуются счетчики `i` и `j`.

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

Вспомогательная строковая переменная `s` позволит нам упростить запись операторов программы.

```
Dim s As String
```

Очищаем окно списка от предыдущих результатов работы программы.

```
lstText.Items.Clear()
```

Оглавление

Вводим количество строк в тексте. Так как количество строк 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 строк"))
Loop Until n > 0
```

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

```
n -= 1
```

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

```
ReDim Tekst(n)
```

Организуем цикл для ввода всех строк текста

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

Вводим очередную строку текста. Обратите внимание, при вводе строки не используется преобразование `Val`. Более того, использование данного преобразования является логической ошибкой, не позволяющей получить правильные результаты.

```
    Tekst(i) = InputBox("Введите " + Str(i) + _
                        "-ю строку текста")
Next
```

В окно списка выводим исходный текст, чтобы пользователь мог контролировать результаты работы программы.

```
For i = 0 To n
    lstText.Items.Add(Tekst(i))
Next
```

Организуем цикл для обработки всех строк текста.

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

Копируем очередную строку во вспомогательную переменную, удаляя при этом пробелы, стоящие перед первым словом строки и за последним.

```
    s = Strings.Trim(Tekst(i))
```

Удаляем из строки `s` лишние пробелы, заменяя два рядом стоящих пробела на один.

```
    Do Until Strings.InStr(s, "  ") = 0
```

Оглавление

```
            s = Strings.Replace(s, "  ", " ")
        Loop
```

С помощью функции `Strings.Split` формируем массив слов обрабатываемой строки.

```
        slova = Strings.Split(s)
```

Определяем номер последнего элемента в массиве слов.

```
        k = UBound(slova)
```

Очищаем вспомогательную переменную, записывая в нее пустую строку.

```
        s = ""
```

Организуем цикл для обработки массива слов. Начальное значение счетчика цикла – ноль, так как нумерация элементов в массиве идет с нуля. Конечное значение – номер последнего слова в массиве слов. Такая организация цикла позволит нам обработать все слова текущей строки текста.

```
        For j = 0 To k
```

Очередное слово строки мы будем записывать во вспомогательную переменную, добавляя его перед ранее сформированной строкой. С помощью пробела будем отделять текущее слово от всей остальной строки. Такой порядок добавления слов позволит нам записать второе слово перед первым, третье перед вторым и так далее. В результате все слова исходной строки окажутся записанными в обратном порядке.

```
            s = slova(j) + " " + s
        Next
```

После завершения внутреннего цикла во вспомогательной переменной записаны все слова исходной строки в обратном порядке. Остается только перенести данные из вспомогательной переменной в исходный массив, удалив при этом из строки пробелы, стоящие перед первым словом и за последним.

```
            Tekst(i) = Strings.Trim(s)
    Next
```

Когда внешний цикл закончит свою работу, в каждой строке слова будут записаны в обратном порядке. Нам остается только вывести измененный массив строк в окно списка. Сначала выводим горизонтальную черту, чтобы зрительно отделить исходные данные от полученных результатов.

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

Так как многострочный текст представляет собой одномерный массив строк, то для его вывода необходимо организовывать цикл.

Оглавление

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

На каждом шаге цикла выводим в окно списка очередную строку текста.

```
    lstText.Items.Add(Tekst(i))
Next
```

Пример работы программы приведен на рис. 7.

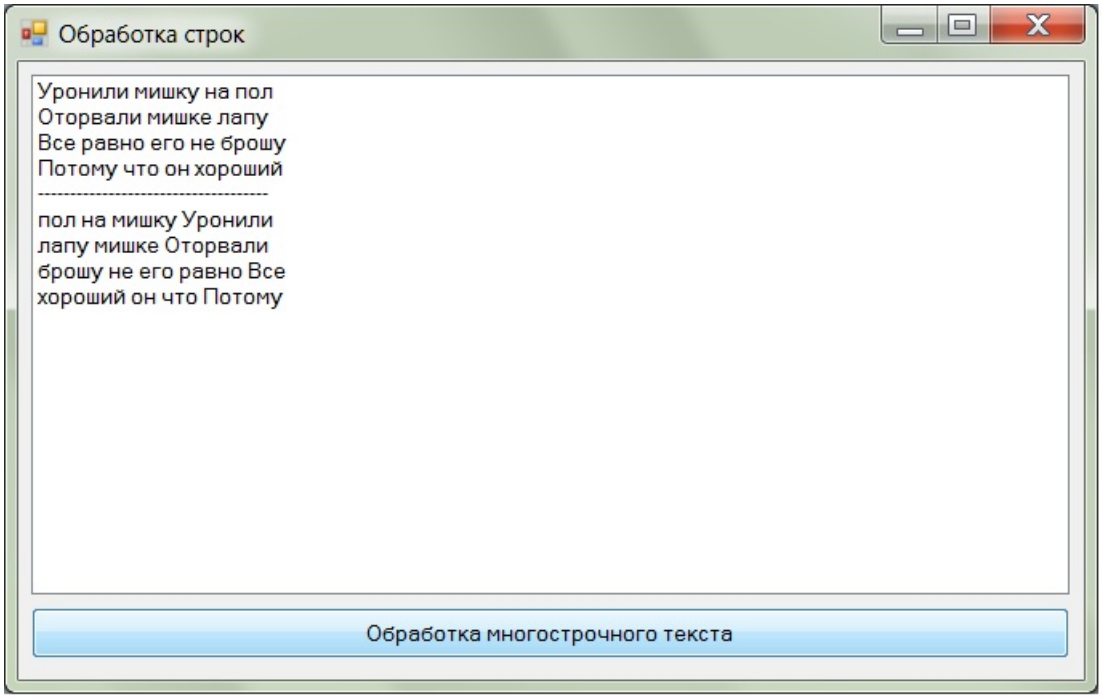

Рис. 7. Пример работы программы обработки многострочного текста

Приложение 1

Дана строка текста, в которой слова разделены произвольным количеством пробелов. Пробелы могут стоять перед первым словом и за последним. Найти самое длинное слово, являющееся палиндромом. Палиндромом называется слово, которое одинаково читается как слева направо, так и справа налево. Исходную строку и полученные результаты вывести в окно списка.

```
Dim s As String
Dim i As Integer
Dim slova() As String
Dim n As Integer
Dim max As String
lstText.Items.Clear()
s = InputBox("Введите строку")
```

Оглавление

```
lstText.Items.Add(s)
s = Strings.Trim(s)
Do Until Strings.InStr(s, "  ") = 0
    s = Strings.Replace(s, "  ", " ")
Loop
slova = Strings.Split(s)
n = UBound(slova)
max = ""
For i = 0 To n
    If slova(i) = Strings.StrReverse(slova(i)) _
       And Len(slova(i)) > Len(max) Then
          max = slova(i)
    End If
Next
lstText.Items.Add("-----------------------------------")
If max <> "" Then
    lstText.Items.Add("Самое длинное слово-палиндром")
    lstText.Items.Add(max)
Else
    lstText.Items.Add("В строке нет слов-палиндромов")
End If
```

Приложение 2

Дана строка текста, в которой слова разделены произвольным количеством пробелов. Пробелы могут стоять перед первым словом и за последним. Найти сумму всех чисел в строке. Исходную строку и полученные результаты вывести в окно списка.

```
Dim s As String
Dim i As Integer
Dim slova() As String
Dim n As Integer
Dim summa, kol As Integer
lstText.Items.Clear()
s = InputBox("Введите строку")
lstText.Items.Add(s)
s = Strings.Trim(s)
```

Оглавление

```
Do Until Strings.InStr(s, "  ") = 0
    s = Strings.Replace(s, "  ", " ")
Loop
slova = Strings.Split(s)
n = UBound(slova)
summa = 0
kol = 0
For i = 0 To n
    If Strings.Trim(Str(Val(slova(i)))) = slova(i) Then
        summa += Val(slova(i))
        kol += 1
    End If
Next
lstText.Items.Add("-----------------------------------")
If kol = 0 Then
    lstText.Items.Add("В строке нет чисел")
Else
    lstText.Items.Add("Сумма =" + Str(summa))
End If
```

Приложение 3

Дана строка текста, в которой слова разделены произвольным количеством пробелов. Пробелы могут стоять перед первым словом и за последним. Упорядочить слова строки по алфавиту. Задачу решить в двух вариантах: с учетом регистра и без учета регистра. Исходную строку и полученные результаты вывести в окно списка.

Сортировка с учетом регистра.

```
Dim s As String
Dim i As Integer
Dim slova() As String
Dim n As Integer
Dim sort As Boolean
Dim z As String
lstText.Items.Clear()
s = InputBox("Введите строку")
lstText.Items.Add(s)
```

Оглавление

```
s = Strings.Trim(s)
Do Until Strings.InStr(s, "  ") = 0
    s = Strings.Replace(s, "  ", " ")
Loop
slova = Strings.Split(s)
n = UBound(slova)
Do
    sort = True
    For i = 0 To n - 1
        If slova(i) > slova(i + 1) Then
            sort = False
            z = slova(i)
            slova(i) = slova(i + 1)
            slova(i + 1) = z
        End If
    Next
Loop Until sort
s = Strings.Join(slova, " ")
lstText.Items.Add("-----------------------------------")
lstText.Items.Add(s)
```

Сортировка без учета регистра.

```
Dim s As String
Dim i As Integer
Dim slova() As String
Dim n As Integer
Dim sort As Boolean
Dim z As String
lstText.Items.Clear()
s = InputBox("Введите строку")
lstText.Items.Add(s)
s = Strings.Trim(s)
Do Until Strings.InStr(s, "  ") = 0
    s = Strings.Replace(s, "  ", " ")
Loop
slova = Strings.Split(s)
```

Оглавление

```
n = UBound(slova)
Do
    sort = True
    For i = 0 To n - 1
        If Strings.UCase(slova(i)) > _
           Strings.UCase(slova(i + 1)) Then
             sort = False
              z = slova(i)
              slova(i) = slova(i + 1)
              slova(i + 1) = z
        End If
    Next
Loop Until sort
s = Strings.Join(slova, " ")
lstText.Items.Add("-----------------------------------")
lstText.Items.Add(s)
```

Список литературы

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

Оглавление