МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

Юго-Западный государственный университет

(ЮЗГУ)

Кафедра вычислительной техники

ОБРАБОТКА СТРОК

Методические указания по выполнению лабораторной работы

для студентов направления подготовки 09.03.01

Курск 2016

УДК 621.3
Составитель: Э.И. Ватутин

Рецензент
Кандидат технических наук, доцент *В.С. Панищев*

Обработка строк: методические указания по выполнению лабораторных работ по дисциплине «Программирование» / Юго-Зап. гос. ун-т; сост.: Э.И. Ватутин; Курск, 2016. 18 с.: табл. 2.

Методические рекомендации содержат сведения по разработке подпрограмм на современных языках программирования высокого уровня.

Предназначены для студентов направления подготовки 09.03.01 «Информатика и вычислительная техника».

Текст печатается в авторской редакции

Подписано в печать _____. Формат 60x84 1/16.
Усл. печ. л. Уч. – изд.л. Тираж 30 экз. Заказ . Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул. 50 лет Октября, 94.

Содержание

Введение

Целью работы является получение практических навыков при разработке программ для обработки строк и символьной информации.

Часто используемым элементом программ являются строки, которые на уровне конструкций программы представляются в виде одномерных массивов символов. Со строками допускается работать так же, как и с массивами, однако их частое использование и выполняемые специфические действия заставляют выделять их в отдельную группу конструкций языка, рассматриваемую в дополнение к массивам. Несомненным достоинством языка Delphi является поддержка строк на уровне компилятора, что зачастую значительно упрощает разработку программ с их использованием.

Прежне всего необходимо ввести понятие кодировок. Достаточно давно, когда емкости оперативной и дисковой памяти были маленькими, а каналы связи – медленными, в компьютерном мире сложилась традиция кодирования символов 8-битным кодом, чему соответствует 256 возможных значений и тип `Char` в Delphi. Существует стандарт ASCII (англ. American Standard Code for Information Interchange), регламентирующий состав первой половины кодовой таблицы (символы с кодами 0…127), что фактически соответствует 7-битной кодировке. Вторая половина кодовой таблицы (символы с кодами 128…255) может подвергаться изменениям для управления символами национальных алфавитов (например, для русского или греческого языка). В операционных системах Microsoft Windows вводится понятие *кодовых страниц* (англ. Code Page, CP) с соответствующими номерами, которые соответствуют различным вариантам расположения символов национальных алфавитов (данные страницы не следует путать со страницами, используемыми при управлении виртуальной памятью в защищенном режиме работы процессора). Например, наиболее распространенными кодовыми страницами с поддержкой кириллицы в Windows являются 866 и 1251 страницы. Символы первой половины кодовой

таблицы для различных кодовых страниц совпадают, а второй – обычно нет. Это приводит к тому, что текст, записанный в виде кодов символов одной кодовой страницы в другой отображается некорректно. Благодаря этому, например, попытка вывода русскоязычных сообщений в консоль с использованием подпрограмм Write/Writeln оканчивается выводом «кракозяб». Аналогичная ситуация может возникнуть, например, при отображении Web-страницы в браузере.

Частично ситуацию спасают подпрограммы `AnsiToOEM` и `OEMToAnsi`, однако при запуске русскоязычной программы, например, в Китае вместо русских букв будут отображены другие символы, т.к. нужная кодовая страница в системе скорее всего будет отсутствовать. Принципиальным выходом из ситуации является использование кодировки UNICODE, в которой под каждый символ отводится 16 бит, что позволяет закодировать 65536 различных символов. В Delphi им соответствует тип `WideChar`. Указанного числа символов с запасом хватает для большинства национальных алфавитов, при этом механизм кодовых страниц фактически становится не нужен, нет несоответствия кодовых страниц и сопутствующих проблем. Однако информация, представленная в данной кодировке, занимает в два раза больше места, что не всегда удобно. Тенденция перехода на кодировку UNICODE наметилась уже более 10 лет назад, однако до сих пор этот переход окончательно не произошел и на практике встречаются символы как 8-, так и 16-битных кодировок, поэтому необходимо уметь работать с применением различных кодировок и при необходимости производить их преобразование.

С целью создания строки символы указанных выше типов могут быть объединены в массив, последующая обработка которого целиком возлагается на программиста:

```
var
  StringAsArray: array [1..100] of Char;
```

Обработка строк в таком виде удобна далеко не всегда, поэтому в Delphi существует поддержка строк на уровне компилятора. Прежде всего она представлена в виде четырех предопределенных типов данных, поддержка которых частично встроена в компилятор:

- `ShortString`;
- `AnsiString`;
- `WideString`;
- `PChar`.

В 80-е годы XX века в наиболее популярных на тот момент языках Паскаль и Си поддержка строк была реализована по разному. Так в Паскале в нулевом байте строки хранилась текущая длина строки, далее следовали символы самой строки, а максимальная длина строки была ограничена 255 символами. Такие строки обычно называются классическими паскалевскими. В языке Си строки хранились иначе: строка с нулевого байта представляла собой массив символов, а ее длина определялась по завершающему символу с кодом 0 и могла превышать 255 символов. Такие строки принято называть нуль-терминированными или ASCIIZ-строками. Разумеется данные типы строк напрямую несовместимы и для их одновременного использования в программе требуется применение дополнительных действий.

Тип `ShortString` в Delphi соответствует классической паскалевской строке, элементы которой имеют тип `Char`, память под строку выделяется статически, а максимальная длина строки ограниченна 255 символами. Сегодня данный тип поддерживается с целью обратной совместимости и имеет весьма ограниченное применение. Он может быть полезен, например, для относительно простого хранения строк фиксированной длины при работе с бинарными файлами. На базе данного статического типа размещения строк в памяти можно объявить т.н. ограниченную строку, максимальная длина которой также ограничена и не может превышать 255 символов:

```
var
  S1: ShortString;  // Максимальная длина строки - 255 символов
  S2: String[5];    // Максимальная длина строки ограничена и равна 5 символам
```

Данные типы строк совместимы между собой и не требуют дополнительных преобразований, однако при их совместном использовании часть информации более длинной строки может бы потеряна

В Паскале тип `String` соответствовал классической паскалевской строке, в Delphi при использовании без указания максимальной длины он по умолчанию соответствует более функциональному типу `AnsiString`, называемому обычно длинными строками (данное поведение может быть изменено с использованием соответствующей опции компилятора «Huge strings» или директивы компилятора `{$H-}`). Строки данного типа размещаются в динамической памяти и их длина теоретически не ограничена, хотя на практике ее максимальное значение обычно ограничено размером сегмента данных программы (почти 2 ГБ). Перед их использованием необходимо выделение памяти с использованием процедуры `SetLength`, вызов которой аналогичен вызову при работе с динамическими массивами, однако зачастую компилятор самостоятельно следит за строками данного типа и при необходимости динамически корректирует длину.

Данный тип строк похож на классические паскалевские строки, только для получения текущей длины строки необходимо использовать функцию `Length`, а не прямое обращение к нулевому байту, что запрещено на уровне компилятора. Строки данного типа завершаются символом с нулевым кодом, что делает возможным их использование совместно с кодом других программных компонентов, ориентированных на использование ASCIIZ-строк (например, DLL-библиотекой на языке Си). Для данного типа строк реализована т.н. copy-on-write семантика, поддерживаемая на уровне компилятора. Так, например, при следующем использовании

```
var
  S1, S2: String;

begin
  S1 := 'ЙЦУКЕНГ';
  S2 := S1;
```

под строку S1 будет выделено 7 байт, а строка S2 на самом деле будет представлять собой ссылку на строку S1, а не полную побайтную копию строки S1, что позволяет более эффективно использовать память из-за отсутствия дублирования данных. При попытке модификации строки S2 будет произведено выделение необходимых 7 байт динамической памяти и настоящее копирование содержимого строки, что дает название используемой стратегии (копирование при записи).

Строки типа `WideString` в целом похожи на `AnsiString` и отличаются от них тем, что используется кодировка UNICODE, а каждый символ такой строки имеет тип `WideChar` и размер 2 байта.

Тип `PChar` соответствует нуль-терминированной строке, управление динамической памятью при его использовании целиком ложится на программиста, что существенно ограничивает сферу его использования в Delphi. Обычно он используется на стыке с программными компонентами, использующими ASCIIZ-строки. Например, при обращении к WinAPI-функции `MessageBox` текстовые параметры должны представлять из себя нуль-терминированные строки, чего можно добиться путем приведения длинных строк к типу `PChar`:

```
uses
  Windows;

var
  S1, S2: String;


begin
  S1 := 'Текст сообщения';
  S2 := 'Заголовок';

  MessageBox(0, PChar(S1), PChar(S2), MB_ICONINFORMATION or MB_OK);
```

При указании в качестве параметров типа `PChar` значений в виде изображений строковых констант в апострофах компилятор автоматически производит необходимое преобразование типа и явное преобразование в виде `PChar('...')` не требуется.

При более детальном рассмотрении тип `PChar` по умолчанию является синонимом типа `PAnsiChar`, наряду с которым существует тип `PWideChar`. Их отличие, как следует из названия, заключается в используемой кодировке.

Для переменных строкового типа определены две операции: *сравнение* и *конкатенация*. При сравнении двух строк допускается использовать любую из шести операций сравнения. Сравнение на равенство и на неравенство реализуется тривиально путем посимвольного сопоставления пары строк. Сравнение на больше и меньше выполняется по следующим правилам:

1. Если код первого символа первой строки меньше кода первого символа второй строки, первая строка считается меньше, чем вторая.
2. Если коды *i*-х символов строк совпадают, производится сравнение кодов $(i+1)$-х символов аналогично предыдущему правилу.
3. Если коды *i*-х символов строк совпадают и одна из строк имеет длину *i* символов, она считается меньше другой, длина которой превосходит *i* символов.

Иногда подобный порядок сравнения называют *лексикографическим*. Рассмотрим применение приведенных правил на примерах.

```
Writeln('a' < 'b');        // Код символа 'a' меньше кода символа 'b'
Writeln('axxx' < 'bxxx');  // Аналогично
Writeln('xxxa' < 'xxxb');  // Первые символы строк 'xxx' совпадают, а за ними идут символы 'a' <
'b'
Writeln('x' < 'xxxb');     // Первые символы строк 'x' совпадают, первая строка короче второй, и,
                           // соответственно, меньше
```

Несложно убедиться в том, что во всех случаях на экран будет выведено значение `True`.

Результат операции конкатенации, обозначаемой символом «+», представляет собой строку, образованную путем слияния двух и более строк. Например, в результате выполнения программы

```
var
```

```
  S1, S2, S3: String;

begin
  S1 := 'First';
  S2 := 'Second';

  S3 := S1 + S2;
```

строка S3 получит значение 'FirstSecond'. Иногда операцию конкатенации называют операцией сложения строк, что некорректно по двум причинам. Во-первых, не существует такого понятия как «сложение строк», в отличие от операции слияния строк. Во-вторых, операция сложения является коммутативной, т.е. допускает изменение порядка слагаемых без изменения значения результата, в то время как операция конкатенации таковой не является. Так, например, результат выражения S2+S1 будет равен 'SecondFirst' и, очевидно, не равен S1+S2.

Кроме операций сравнения и конкатенации существует ряд подпрограмм стандартной библиотеки, облегчающих типовые операции при обработке строк.

Таблица 1

Подпрограммы для работы со строками

Подпрограмма	Параметры	Результат	Назначение
Length	S: String	Integer	Возвращает текущую длину строки
SetLength	S: String; L: Integer	–	Изменяет длину строки S, делая ее равной L символам
Pos	SubS: String; S: String	Integer	Возвращает позицию первого вхождения подстроки SubS в строку S или 0, если подстрока не найдена
PosEx	SubS: String; S: String; From: Integer	Integer	Действие аналогично предыдущей функции за исключением того, что поиск подстроки начинается с символа From
Copy	S: String; From: Integer; Count: Integer	String	Копирует подстроку строки S из Count символов начиная с символа From

Delete	S: String; From: Integer; Count: Integer	–	Удаляет из строки S подстроку из Count символов, начиная с символа From
Insert	SubS: String; DstS: String; From: Integer	–	Вставляет подстроку SubS в строку DstS, начиная с символа From
LowerCase	S: String	String	Переводит строку в нижний регистр
UpperCase	S: String	String	Переводит строку в верхний регистр

Большинство подпрограмм для работы со строками объявлено в модулях System.pas, SysUtils.pas и StrUtils.pas. Рассмотрим несколько поясняющих примеров использования указанных подпрограмм.

```
S := 'bbaabbccaa';

Writeln(Pos('aa', S));       // 3

Writeln(PosEx('aa', S, 4));  // 9

Writeln(Copy(S, 5, 6));      // 'bbccaa'

Delete(S, 2, 3);
Writeln(S);                  // 'bbbccaa'

Insert('xx', S, 2);
Writeln(S);                  // 'bxxbbccaa'

Writeln(UpperCase(S));       // 'BXXBBCCAA'
```

Если функциональность стандартных подпрограмм чем-либо не устраивает, можно разработать свои аналоги, в которых допускается как использование стандартных подпрограмм, так и посимвольная обработка строк.

Обработка строк часто вплотную соприкасается с еще одним достаточно удобным элементом языка Delphi – множествами.

Пример программы

Задача. Проверить корректность ввода числа в формате с плавающей точкой.

Число в формате с плавающей точкой в общем случае имеет следующие составляющие:

[Знак] Мантисса [[e|E] [Знак] Порядок],

причем знак может принимать значения «+» или «−», мантисса представляет собой число с фиксированной точкой, а порядок – целое число. Приведенная запись соответствует формальному грамматическому определению числа с плавающей точкой в современных языках программирования и является одной из форм записи т.н. *форм Бэкуса-Наура*, при этом необязательные элементы (например, знак мантиссы) взяты в квадратные скобки, а элементы по выбору отделены друг от друга символом «|». В таблице 2 приведены примеры строк, соответствующих и не соответствующих приведенной выше записи.

Таблица 2

Примеры корректных и некорректных вещественных чисел

Корректные вещественные числа	**Некорректные вещественные числа**
45 +265 −23 456.548 +3214.32e−45	r45At +2.6.1 .+3456 −7+4 E456.548 456.548E 0.314e1.1

Проверка правильности записи вещественного числа «в лоб» может быть достаточно громоздкой, поэтому разобьем ее на элементарные действия. Прежде всего, реализуем функцию для проверки того, что

введенное строковое значение является корректным целым числом без знака. В состав подобного числа могут входить только цифры.

```
function IsCorrectUnsignedInt(S: String): Boolean;
var
  I: Integer;
begin
  Result := False;

  if S = '' then
    exit;

  for I := 1 to Length(S) do
    if not (S[I] in ['0'..'9']) then
      exit;

  Result := True;
end;
```

На базе разработанной функции можно реализовать функцию, проверяющую корректность целого числа со знаком. Ее отличие от предыдущей функции заключается в том, что на первой позиции в строке допускается присутствие знака «+» или «−».

```
function IsCorrectSignedInt(S: String): Boolean;
begin
  if (Length(S) > 0) and (S[1] in ['+', '-']) then
    Result := IsCorrectUnsignedInt(Copy(S, 2, Length(S)-1))
  else
    Result := IsCorrectUnsignedInt(S);
end;
```

На базе разработанных функций можно реализовать функцию проверки корректности ввода числа с фиксированной точкой. Можно заметить, что при удалении точки из числа с фиксированной точкой оставшаяся строка должна представлять из себя корректное целое число, на чем основана приведенная ниже проверка. При этом из правила есть исключение для некорректных строк вида «.+5», для обработки которых необходимо отдельное условие.

```
function IsCorrectFixedFloat(S: String): Boolean;
var
  I: Integer;
begin
  // Обработка ситуации вроде ".+5"
  if (Length(S) > 1) and (S[1] = '.') and (S[2] in ['+', '-']) then begin
    Result := False;
    exit;
  end;

  I := Pos('.', S);
  if I <> 0 then
```

```
    Delete(S, I, 1);
  Result := IsCorrectSignedInt(S);
end;
```

На базе разработанных функций можно реализовать искомую функцию проверки на корректность числа в формате с плавающей точкой. Для этого прежде всего в числе производится поиск буквы «E» (чтобы не организовывать поиск два раза для заглавной и строчной букв строка предварительно преобразуется в верхний регистр). В случае, если искомая буква не найдена, число представляет собой форму записи с фиксированной точкой. Если же буква «E» присутствует в записи, то слева от нее должна располагаться подстрока с числом с фиксированной точкой (мантисса), а справа – подстрока с целым числом со знаком (порядок). Учитывая, что выше были разработаны соответствующие функции проверки на корректность, искомая функция может быть представлена в следующем виде:

```
function IsCorrectFloatingPoint(S: String): Boolean;
var
  I: Integer;
begin
  S := UpperCase(S);
  I := Pos('E', S);

  if I = 0 then
    Result := IsCorrectFixedFloat(S)
  else
    Result := IsCorrectFixedFloat(Copy(S, 1, I-1)) and
              IsCorrectSignedInt(Copy(S, I+1, Length(S)-I));
end;
```

Приведенное решение основано на декомпозиции поставленной задачи на частные элементарные подзадачи в соответствии с рассмотренным выше процедурным подходом. Для тестирования разработанных подпрограмм с учетом приведенных выше (табл. 2) тестовых примеров может быть использована следующая программа:

```
var
  Tests: array [1..13] of String = (
      '45',
      '+265',
      '-23',
      '456.548',
      '+3214.32e-45',
      '',
      'r45At',
      '+2.6.1',
      '.+3456',
```

```
      '-7+4',
      'E456.548',
      '456.548E',
      '0.314e1.1'
  );
  I: Integer;

begin
  for I := Low(Tests) to High(Tests) do
    Writeln('"', Tests[I], '" - ', Integer(IsCorrectFloatingPoint(Tests[I])));
  Readln;
end.
```

Индивидуальные задания

1. Дана строка *C*, содержащая текст. Сформировать из строки *C* новую строку путем перестановки слов текста в обратной последовательности.

2. Задана строка *S*, содержащая набор слов, разделенных пробелами. Удалить из строки все слова, содержащие заданную подстроку.

3. Дан массив строк, содержащих слова, разделенные пробелами. Упорядочить массив по убыванию средней длины слов.

4. Написать программу, которая инвертирует порядок букв в словах заданного предложения. После преобразования порядок слов должен остаться прежним.

5. Дано предложение из *N* слов. Сформировать из него новое предложение путем циклического сдвига слов вправо на *i* позиций. Например, для строки «обработка строк удобна с использованием стандартных подпрограмм» и $i = 2$ должна быть выдана строка «стандартных подпрограмм обработка строк удобна с использованием».

6. Составить программу для синтаксического анализа строки, содержащей число в формате с плавающей запятой. Результатом анализа должны быть: признак корректности и класс числа (целое без знака, целое со знаком, вещественное с фиксированной точной, вещественное с плавающей точкой) в случае корректности исходной строки.

7. Разработать программу, которая выделяет в заданной строке самую длинную подстроку из одинаковых элементов.

8. Дан массив слов и строка текста. Проверить строку на наличие слов, не входящих в состав массива.

9. Проверить, является ли заданная строка корректным идентификатором, записанным по правилам языка Delphi.

10. Произвести разбиение заданного предложения на слова. Учесть, что в составе предложения могут быть знаки препинания, в составе слов – дефисы, а слова могут отделяться друг от друга более чем одним пробелом.

11. Найти в заданном предложении самый длинный палиндром (зеркально-симметричное слово).

12. В заданном предложении выделить самое длинное и самое короткое слово.

13. Составить программу анализа арифметического выражения на правильность расстановки скобок. Учесть: а) случай использования скобок одного типа (например, «`1 + x*(3+sin(y))`»); б) случай использования скобок разных типов (например, «`[x*(y-1) + y*(x-1)]*2`»).

14. Преобразовать множество (set of byte) в строковое представление вида «`{a1, a3, a10}`». По возможности оформить преобразование в виде подпрограммы.

15. Преобразовать строку вида «`{a1, a3, a10}`» в множество (set of byte). Оформить преобразование в виде подпрограммы.

16. Подсчитать количество слов в строке. Учесть возможность наличия в строке знаков препинания.

17. Строка представляет собой фрагмент текста (несколько абзацев). Заменить все многократные вхождения пробелов на одиночные пробелы («_ _ … _» → «_») без изменения остальных символов.

18. Дано слово *S* и массив слов *A*. Вывести на экран все слова массива *A*, которые можно составить из букв слова *S*. Например, из слова «процедура» можно составить слова «пруд», «руда», «цедра» и нельзя слова «ара» и «цена».

19. Задан словарь (массив слов) и номер телефона. Найти и вывести на экран все слова, с помощью которых можно закодировать заданный номер телефона. Например, номер телефона 533-232-72 можно представить словом «незадача», а 64-55-72 – «Тимоха».
20. Заданы две строки, каждая из которых является строковым представлением числа в формате с фиксированной точкой. Найти сумму чисел. В случае наличия ошибок вывести соответствующее сообщение. Функции `FloatToStr()`, `FloatToStrF()`, `Format()`, `IntToStr()` не использовать!

Содержание отчета

1. Титульный лист.
2. Индивидуальное задание.
3. Краткое описание стратегии решения.
4. Листинг программы.
5. Тестовые примеры, результаты тестирования.
6. Выводы.

Контрольные вопросы

1. Для чего применяются строки?
2. Как выбор кодировки влияет на двоичное содержимое строки?
3. Какие подпрограммы предназначены для обработки строк?

Библиографический список

1. Емельянов С.Г., Ватутин Э.И., Панищев В.С., Титов В.С. Процедурно-модульное программирование на Delphi: учебное пособие. М.: Аргамак-Медиа, 2014. 352 с.
2. Зотов И.В., Ватутин Э.И., Борзов Д.Б. Процедурно-ориентированное программирование на C++: учебное пособие. Курск: КурскГТУ, 2008. 211 с.