Символы и строки.

Процедуры и функции работы со строками.

Записи.

Строковый тип данных

Для обработки строковой информации в Турбо Паскаль введен строковый тип данных.

Строкой в Паскале называется последовательность из определенного количества символов.

Количество символов последовательности называется длиной строки. Синтаксис:

var s: string[n];

var s: string;

n - максимально возможная длина строки - целое число в диапазоне 1..255.

Если этот параметр опущен, то по умолчанию он принимается равным 255.

Строковые константы записываются как последовательности символов, ограниченные апострофами. Пример:

'Текстовая строка'

Допускается формирование строк с использованием записи символов по десятичному коду (в виде комбинации # и кода символа)

#54#32#61

и управляющих символов (комбинации ^ и некоторых заглавных латинских букв).

'abcde'^A^M

Пустой символ обозначается двумя подряд стоящими апострофами. Если апостроф входит в строку как литера, то при записи он удваивается.

Переменные, описанные как строковые с разными максимальными длинами, можно присваивать друг другу, хотя при попытке присвоить короткой переменной длинную лишние символы будут отброшены.

Выражения типа **char** можно присваивать любым строковым переменным.

В Турбо Паскаль имеется простой доступ к отдельным символам строковой переменной: **i-й символ** переменной **st** записывается как **st[i].**

Например, если **st** - это **'Строка'**, то

st[1] - это 'С', st[2] - это 'т', st[3] - 'р' и так далее.

Над строковыми данными определена операция слияния (конкатенации), обозначаемая знаком **+.**

Например:

```
a := 'Turbo';
b := 'Pascal';
c := a + b;
```

В этом примере переменная **c** приобретет значение **'TurboPascal'.**

Кроме операции слияния (конкатенации) над строками определены операции сравнения <,>,=,<>,<=,>=.

Две строки сравниваются посимвольно, слева направо, по кодам символов. Если одна строка меньше другой по длине, недостающие символы короткой строки заменяются символом с кодом 0.

Процедуры и функции для работы со строками

В системе Turbo Pascal имеется несколько полезных стандартных процедур и функций, ориентированных на работу со строками.

Length(s:string):integer

Функция возвращает в качестве результата значение текущей длины строки-параметра

Пример.

```
n := length('Pascal'); {n будет равно 6}
```

Concat(s1,[s2,...,sn]:string):string

Функция выполняет слияние строк-параметров, которых может быть произвольное количество. Каждый параметр является выражением строкового типа. Если длина строки-результата превышает 255 символов, то она усекается до 255 символов. Данная функция эквивалентна операции конкатенации "+" и работает немного менее эффективно, чем эта операция.

Copy(s:string; index:integer; count:integer):string

Функция возвращает подстроку, выделенную из исходной строки s, длиной **count** символов, начиная с символа под номером **index.**

Пример.

```
s := 'Система Turbo Pascal';
s2 := copy(s, 1, 7); {s2 будет равно 'Система'}
s3 := copy(s, 9, 5); {s3 будет равно 'Turbo'}
s4 := copy(s, 15, 6); {s4 будет равно 'Pascal'}
```

Delete(var s:string; index,count:integer)

Процедура удаляет из строки-параметра **s** подстроку длиной **count** символов, начиная с символа под номером **index.**

Пример.

```
s := 'Система Turbo Pascal';
delete(s,8,6); {s будет равно 'Система Pascal'}
```

Insert(source:string; var s:string;index:integer)

Процедура предназначена для вставки строки **source** в строку **s,** начиная с символа **index** этой строки.

Пример.

```
s := 'Система Pascal';
insert('Turbo ',s,9); {s будет равно 'Система Turbo Pascal'}
```

Pos(substr,s:string):byte

Функция производит поиск в строке **s** подстроки **substr.** Результатом функции является номер первой позиции подстроки в исходной строке. Если подстрока не найдена, то функция возвращает 0.

Пример.

```
s := 'Система Turbo Pascal';
x1 := pos('Pascal', s); {x1 будет равно 15}
x2 := pos('Basic', s);   {x2 будет равно 0}
```

Str(X: *арифметическое выражение*; var st: string)

Процедура преобразует численное выражение **X** в его строковое представление и помещает результат в **st.**

Val(st: string; x: *числовая переменная*; var code: integer)

Процедура преобразует строковую запись числа, содержащуюся в **st,** в числовое представление, помещая результат в **x.**

x - может быть как целой, так и действительной переменной. Если в **st** встречается недопустимый (с точки зрения правил записи чисел) символ, то преобразование не происходит, а в **code** записывается позиция первого недопустимого символа.

Выполнение программы при этом не прерывается, диагностика не выдается. Если после выполнения процедуры **code** равно 0, то это свидетельствует об успешно произошедшем преобразовании.

Некоторые функции, связанные с типом **char,** достаточно часто используются и при работе со строками.

Chr(n: byte): char

Функция возвращает символ по коду, равному значению выражения **n**. Если **n** можно представить как числовую константу, то можно также пользоваться записью **#n.**

Ord(ch: char): byte;

В данном случае функция возвращает код символа **ch.**

UpCase(c: char): char;

Если **c -** строчная латинская буква, то функция возвращает соответствующую прописную латинскую букву, в противном случае символ с возвращается без изменения.

Примеры заданий:

1. Определить и вывести на экран длину введенной пользователем строковой величины.

```
Program Str1;
    Var
    S : String;
    Begin
Writeln('Введите последовательность символов');
    Readln(S);
    Writeln('Вы ввели строку из ',Length(S), '
    символов');
End.
```

2. Введенную строку вывести на экран по одному символу в строке экрана.

```
Program Str2;
    Var
   S : String;
    I : Byte;
Begin
Writeln('Введите строку');
    Readln(S);
    For I:=1 to Length(S) do
    Writeln(S[I]);
End.
```

3. Вывести на экран кодовую таблицу.

```
Program Str3;
    Var
    I : Byte;
    Begin
For I:=32 to 255 do
    Write('VV',I:4, '-',Chr(I))
End.
```

Цикл в программе начинается с 32 потому, что символы с кодами от 0 до 31 являются управляющими и не имеют соответствующего графического представления.

4. **Определить, является ли введенная строка "перевертышем". Перевертышем называется такая строка, которая одинаково читается с начала и с конца. Например, "казак" и "потоп" - перевертыши, "канат" - не перевертыш".**

Решение:

1. из введенной строки сформируем другую строку из символов первой, записанных в обратном порядке.
2. сравним первую строку со второй; если они окажутся равны, то ответ положительный, иначе - отрицательный.

```
Program Str4;
    Var
    S,B : String;
    I : Byte;
    Begin
     Writeln('Введите строку');
    Readln(S);
    B:=''; {Переменной B присваиваем значение
    "пустая строка"}
    For I:=1 to Length(S) do
    B:=S[I]+B; {Конкатенация. Символы строки S
    пристыковываются к переменной B слева. Самым
    левым окажется последний.}
    If B=S Then Writeln('Перевертыш') Else Writeln('Не
    перевертыш')
End.
```

5. Найти сумму цифр введенного натурального числа.

```
Program Str5;
    Var
   S : String;
    I,X,A,C : Integer;
Begin
Writeln('Введите натуральное число');
    Readln(S); {Число вводится в строковую переменную}
    A:=0;
    For I:=1 To Length(S) Do
    Begin
Val(S[I],X,C); {Цифровой символ превращается в число}
    A:=A+X {Цифры суммируются}
End;
    Writeln('Сумма цифр равна ',A)
End.
```

6. Во введенной строке заменить все вхождения подстроки 'ABC' на подстроки 'KLMNO'".

```
Program Str6;
    Var
   S : String;
    A: Byte;
Begin
Writeln('Введите строку');
    Readln(S);
    While Pos('ABC',S)<>0 Do
    Begin
A:= Pos('ABC',S);
    Delete(S,A,3);
    Insert('KLMNO',S,A)
End;
    Writeln(S)
End.
```

Записи

Запись представляет собой совокупность ограниченного числа логически связанных компонент, принадлежащих к разным типам. Компоненты записи называются полями, каждое из которых определяется именем. Поле записи содержит имя поля, вслед за которым через двоеточие указывается тип этого поля. Поля записи могут относиться к любому типу, допустимому в языке Паскаль, за исключением файлового типа.

Описание записи в языке Паскаль осуществляется с помощью служебного слова **record,** вслед за которым описываются компоненты записи. Завершается описание записи служебным словом **end**

Например, телефонный справочник содержит фамилии и номера телефонов, поэтому отдельную строку в таком справочнике удобно представить в виде следующей записи:

```
type
TRec = Record
FIO: String[20];
TEL: String[7]
end;
var
rec: TRec;
```

Описание записей возможно и без использования имени типа, например:

```
var
rec: Record
FIO: String[20];
TEL: String[7]
end;
```

Обращение к записи в целом допускается только в операторах присваивания, где слева и справа от знака присваивания используются имена записей одинакового типа. Во всех остальных случаях оперируют отдельными полями записей.

Чтобы обратиться к отдельной компоненте записи, необходимо задать имя записи и через точку указать имя нужного поля, например:

rec.FIO, rec.TEL

Такое имя называется **составным**. Компонентой записи может быть также запись, в таком случае составное имя будет содержать не два, а большее количество имен.

Обращение к компонентам записей можно упростить, если воспользоваться оператором присоединения **with.**

Он позволяет заменить составные имена, характеризующие каждое поле, просто на имена полей, а имя записи определить в операторе присоединения:

with rec do *оператор*; Здесь rec - имя записи, *оператор* - оператор, простой или составной. *Оператор* представляет собой область действия оператора присоединения, в пределах которой можно не использовать составные имена. Например для нашего случая:

```
with rec do
begin
FIO:='Иванов А.А.';
TEL:='2223322';
end;
```

Это полностью идентично следующему:

```
rec.FIO:='Иванов А.А.';
rec.TEL:='2223322';
```

Инициализация записей может производиться с помощью типизированных констант:

```
Program zz;
type   RecType = Record
 x,y: Word;
ch: Char;
dim: Array[1..3] of Byte
end;
const
Rec: RecType = ( x: 127; y: 255; ch: 'A'; dim: (2, 4, 8) );
begin
Writeln(rec.x,rec.y,rec.ch,rec.dim[3]) ;
End.
```

Особой разновидностью записей являются записи с вариантами, которые объявляются с использованием зарезервированного слова **case.** С помощью записей с вариантами вы можете одновременно сохранять различные структуры данных, которые имеют большую общую часть, одинаковую во все структурах, и некоторые небольшие отличающиеся части.

Например, сконструируем запись, в которой мы будем хранить данные о некоторой геометрической фигуре (отрезок, треугольник, окружность).

```
typeTFigure = record
type_of_figure: string[10];
color_of_figure: byte;
  ...  case integer of
1: (x1,y1,x2,y2: integer);
2: (a1,a2,b1,b2,c1,c2: integer);
3: (x,y: integer; radius: word);
end;
var figure: TFigure;
```

Таким образом, в переменной **figure** мы можем хранить данные как об отрезке, так и о треугольнике или окружности. Надо лишь в зависимости от типа фигуры обращаться к соответствующим полям записи.

Индивидуальные поля для каждого из типов фигур занимают тем не менее одно адресное пространство памяти, а это означает, что одновременное их использование невозможно.

В любой записи может быть только одна вариантная часть. После окончания вариантной части в записи не могут появляться никакие другие поля. Имена полей должны быть уникальными в пределах той записи, где они объявлены.