Символьные переменные и строки. Организация, размещение в памяти, процедуры и функции обработки строк. Множества и записи

Лекция12

В Pascal имеется два типа данных для работы с текстами:

- **char** – литерный или символьный тип;
- **string** – строковый тип или просто строка.

*Символьный тип

Значением переменных символьного типа **char** является один символ. Каждому символу соответствует *код символа* — целое число в диапазоне от 0 до 255. Символьный тип является порядковым.

Над данными символьного типа определены следующие *операции отношения*: **=, <>,** <, >, <=, >=, вырабатывающие результат логического типа.

При выполнении всех операций сравнения коды символов сравниваются как обычные целые числа. При этом получается, что любая заглавная буква всегда меньше соответствующей ей строчной, т. к. в кодовой таблице сначала располагаются все заглавные буквы, а затем строчные. Точно так же любая буква латинского алфавита всегда меньше любой буквы русского алфавита.

Для данных символьного типа определены следующие стандартные функции:

- **chr (x)** – возвращает значение символа по его коду;
- **ord(ch)** – возвращает код заданного символа **ch;**
- **pred(ch)** – возвращает предыдущий символ;
- **succ(ch)** – возвращает следующий символ;
- **upcase(ch)** – преобразует строчную букву в заглавную. Обрабатывает буквы только латинского алфавита.

Например:

ord(’A’)=65

chr(128)= ’Б’

pred(’Б’)= ’A’

succ(‘Г’)=’Д’

upcase(‘n’)= ’N’

*Строковый тип

Строка — это последовательность символов. Максимальное количество символов в строке *(длина строки)* может изменяться от 1 до 255.

Строковую переменную можно задать следующим образом:

var Идентификатор: string[макс. длина строки];

Например:

```
var  fm:string[20], st:string;
```

Если максимальная длина строки не указывается, то она равна 255 байт.

Организация строк, размещение в памяти

Строка в языке Pascal трактуется одновременно и как скалярное (простое) значение, и как массив символов. Поэтому некоторые операции могут выполняться над строкой целиком (например, ввод и вывод), но одновременно можно обрабатывать и каждый символ по отдельности. В этом отличие строки от массива. Способ внутреннего представления строки тоже немного отличается от внутреннего представления массива.

Для строки из n символов в памяти отводится n+1 байт: n байт — для хранения символов строки и один дополнительный байт — для значения текущей длины строки. Этот дополнительный байт имеет номер 0, соответственно первый символ строки имеет номер 1, второй — номер 2 и т. д.

Из этого представления становится понятным и ограничение на максимальную длину строки — 255 символов, т. к. для хранения длины строки отведен всего один байт.

К любому символу в строке можно обратиться как к элементу одномерного массива по номеру данного символа в строке.

В отличие от массивов переменные строкового типа могут участвовать как операнды в инструкциях ввода/вывода.

При вводе строки количество символов в ней определяется автоматически, при этом автоматически заполняется нулевой байт.

Над строковыми данными допустимы *операция сцепления* (конкатенации) и *операции отношения*.

Операции =, <>, >, <, >=, <= выполняют сравнение двух строковых операндов. Сравнение строк производится слева направо до первого несовпадающего символа, и та строка считается большей, в которой код первого несовпадающего символа больше. Строки считаются равными, если они совпадают по длине и содержат одни и те же символы.

*В*се строковые типы считаются *совместимыми по присваиванию*, т. е. можно присваивать одно другому значения строк различного размера. Размеры строк при этом не контролируются. Если значение переменной после выполнения оператора присваивания превышает по длине максимальный размер, указанный при объявлении, *все лишние символы справа отбрасываются*, что служит источником трудно находимых ошибок.

Процедуры и функции работы со строками

- **Функция length (st)** — вычисляет текущую длину строки **st.**

Пример. Пользователь вводит слово. Вывести количество букв в этом слове.

```
  var s:string[20]; kol:integer;
begin
writeln('Vvedite slovo');
readln(s);
kol:=length(s);
writeln('V slove ', kol,' bukv');
readln;
end.
```

- **Функция copy (st, poz, n)** — выделяет из строки **st** подстроку длиной n символов, начиная с позиции **poz.** Если **poz** больше длины строки, то результатом будет пустая строка.

Пример. Пользователь вводит слово. Выведите на экран третью и четвертую буквы этого слова.

```
ar s,s1:string[20];
begin
writeln('Vvedite slovo');
readln(s);
s1:=copy(s,3,2);
writeln(s1);
readln;
end.
```

- **Функция concat(str1,str2,...,strn)** — выполняет сцепление строк **str1, str2, strn** в том порядке, в каком они указаны в списке параметров.

Пример. Пользователь вводит фамилию учащегося и номер группы. Вывести сообщение, что он учится в этой группе. Это сообщение должно храниться в одной переменной.

```
var sf,sg,s:string;
begin
writeln('Vvedite fio');
readln(sf);
writeln('Vvedite nomer grup');
readln(sg);
s:=concat(sf,' uchitsja  v gr. nomer ',sg);
writeln(s);
readln; end.
```

- **Функция pos (str1, str2)** — обнаруживает первое появление в строке **str2** подстроки **str1.** Результат имеет целочисленный тип и равен номеру той позиции, в которой находится первый символ подстроки **str1.** Если в **str2** подстроки **str1** не найдено, результат равен 0.

Пример. Пользователь вводит предложение. Выбросить из него первое слово.

```
var s1,s:string; p:integer;
begin
writeln('Vvedite predlojenie');
readln(s);
p:=pos(' ',s);
s1:=copy(s,p+1,length(s)-p);
writeln(s1);
readln;
end.
```

- **Процедура delete(st, poz, n)** – удаление n символов строки **st,** начиная с позиции **poz.** Если значение **poz** больше, чем размер строки, ничего не удаляется.

Пример. Пользователь вводит слово. Удалить из него первые две буквы.

```
var s:string;
begin
writeln('Vvedite slovo');
readln(s);
delete(s,1,2);
writeln(s);
readln;
end.
```

Процедура insert (str1, str2, poz) — вставка строки **str1** в строку **str2**, начиная с позиции **poz.**

Пример. Пользователь вводит текст, основное слово и слово-дополнение. Вставить во введенном тексте после основного слова слово-дополнение.

```
var t,so,sd:string; p:integer;
begin
writeln('Vvedite text');   readln(t);
writeln('Vvedite osnovnoe slovo');    readln(so);
writeln('Vvedite slovo-dopolnenie');   readln(sd);
p:=pos(so,t);
insert(' '+sd,t,p+length(so));
writeln(t);
readln;
end.
```

Pascal позволяет преобразовывать данные числового типа в строку символов, а также преобразовывать строку в число, если она содержит последовательность символов, удовлетворяющую правилам записи чисел. Для этой цели имеются две процедуры: **str** и **val.**

Процедуру **str** удобно использовать для вставки числовых данных в какой-либо текст, т. к. операция конкатенации и процедура **insert** могут работать только со строковыми данными.

Вызов этой процедуры имеет вид:

str (number, st) — преобразование числового значения величины **number** в строку **st.** После **number** может записываться формат, аналогичный формату вывода. Если в формате указано недостаточное для вывода количество разрядов, поле вывода расширяется автоматически до нужной длины.

Например:

```
Var
    sl, s2, s3:  string;
    num1 : integer;
    num2: real;
begin
    num1:=5;
    num2:=7.75;
    str(num1, s1);
    str(num2, s2);
    str(num2: 3 :1, s3);
```

В результате:

s1='5'; s2= ‘7.75000000000E+00 ' ; s3= '7.8 '

Процедура **val** часто используется, чтобы преобразовать введенную с клавиатуры или прочитанную из файла строку в числовые данные.

val (st, number, code) — преобразует значение **st** в величину целочисленного или вещественного типа и помещает результат в **number. code** — целочисленная переменная. Если во время операции преобразования ошибки не обнаружено, то значение **code** равно нулю, а если ошибка обнаружена (строковое значение не переводится в цифровое), **code** будет содержать номер позиции первого ошибочного символа, а значение **number** не определено.

Например:

```
s1:= '5.78'; s2:= '5,78';
val(s1, num1, cod1);
val(s2,num2,cod2);
```

в результате cod1=0, cod2=2 - второй символ ошибочный

Использование функции **val** позволяет избежать ошибки выполнения **Invalid numeric format,** возникающей при неправильном вводе числовых данных. Приходится вводить вместо числовой переменной строковую (в ней разрешены любые символы). После этого введенная строка преобразуется в число и, если преобразование невозможно, об этом выводится сообщение на русском языке. Чаще всего в таких случаях не прерывают программу, а просят пользователя повторить ввод. Здесь очень удобно использовать цикл **repeat**. Например:

```
var s:string; n, error: integer;
begin
repeat
write('Введите число ');  readln(s);  val(s, n, error);
if error>0 then writeln('Неверный символ № ',error)
until error=0;
```

*Множества

Множество — это набор элементов одинакового типа, которые рассматриваются как единое целое. Элементы множества не пронумерованы, следовательно, *нельзя обратиться к отдельному элементу множества по его индексу.*

Тип элементов множества называется *базовым типом* множества. *Область значений типа множества* — набор всевозможных подмножеств, составленных из элементов базового типа.

В языке Pascal имеются ограничения на базовый тип. Это может быть только порядковый тип, количество значений которого не превышает 256. Из простых типов к таким относятся **char, byte, boolean.** Разрешается использовать перечисляемый тип и диапазон (если он включает не больше 256 элементов).

Количество элементов множества иногда называют *мощностью* данного множества.

Для объявления множественного типа используется словосочетание **set of**.

var ИмяПеременной: set of Тип;

Например: mn: set of byte;

В памяти множества представлены особым образом. Каждому значению базового типа множества в памяти отводится 1 бит (не байт!). Следовательно, максимальный размер ячейки памяти, отводимой под множество, составляет 32 байта. Поскольку все значения порядкового типа расположены строго по порядку, 1 в соответствующем бите означает наличие данного значения в множественной переменной, а 0 — отсутствие.

Исходя из особенностей внутреннего представления множеств, можно сделать два основных вывода:

- *в множестве не может быть одинаковых элементов,* что согласуется и с нашими математическими понятиями;
- все операции над множествами выполняются значительно эффективней, чем над другими структурами данных.

При работе с множествами допускается использование следующих операций:

- отношения (=, **<>, >=, <=);**
- объединения множеств (+);
- пересечения множеств (*);
- разности множеств (-);
- проверки принадлежности элемента множеству (in).

Операция "больше или равно" (>=). Эта операция используется для определения принадлежности одного множества другому. Результат операции A>=B равен **true,** если все элементы множества B содержатся в множестве A. В противном случае результат равен **false**.

Операция **in.** Эта операция используется для проверки принадлежности какого-либо значения указанному множеству. Она обычно применяется в условных операторах.

Объединение множеств (+). Объединением двух множеств является третье множество, содержащее элементы обоих множеств.

Пересечение множеств (*). Пересечением двух множеств является третье множество, которое содержит элементы, входящие одновременно в оба множества.

Разность множеств (-). Разностью двух множеств является третье множество, которое содержит элементы первого множества, не входящие во второе множество.

Элемент множества берется в квадратные скобки. Пустое множество обозначается в Паскале [].

В Pascal отсутствуют средства ввода/вывода элементов множества.

Тем не менее, не трудно написать процедуру для вывода элементов множества. Например, процедура для вывода множества символов может иметь следующий вид:

```
type charset=set of char;
procedure writeset(a:charset) ;
 var c:char;
 begin
for c:=chr(0) to chr(255) do
        if c in a then write(c,'  ');
 writeln;
end;
```

Пользователь вводит два слова. Определить какие из букв второго слова содержатся в первом.

```
type mnoj=set of char;
var   a:mnoj;  i:integer; st1,st2:string[25];
begin
   writeln('Vvedite pervoe slovo');     readln(st1);
   writeln('Vvedite vtoroe slovo');      readln(st2);
   a:=[];
   for i:=1 to length(st1) do  a:=a+[st1[i]];
   for i:=1 to length(st2) do
      if st2[i] in a then
      begin
         writeln(st2[i],' ');
         a:=a-[st2[i]];
      end;
    readln;   end.
```

*Записи

Запись — это структурированный тип данных, состоящий из фиксированного числа компонентов одного или нескольких типов, называемых *полями* записи. Компоненты записи могут быть разного типа. Чтобы можно было ссылаться на тот или иной компонент записи, *каждое ее поле имеет свое имя* (а не номер, как элемент массива).

Записи можно объявить следующим способом.

```
type
ИмяТипа = record
ИмяПоля1: ТипПоля1;
ИмяПоля2: ТипПоля2;
…
ИмяПоляN: ТипПоляN
end;
```

Затем объявляются переменные соответствующего типа.

var

ИмяПеременной : ИмяТипа ;

Например:

```
type avto=record
        number: integer;      { номер автомобиля }
        marka: string[20];  { марка автомобиля }
        fio: string[40];      { фамилия, инициалы владельца }
        address: string[60]{ адрес владельца }
end;
var m,v: avto;
```

Значения полей записи могут использоваться в выражениях. *Обращение к значению поля* осуществляется с помощью конструкции, содержащей имя переменной и имя поля, разделенные точкой. Такая комбинация называется *составным именем*. Например, для того чтобы получить доступ к полям записи **avto,** надо записать **m.number, m.marka, m.fio, m.address.**

Составное имя можно использовать везде, где допустимо применение типа его поля. Для присваивания полям значений используется оператор присваивания.

Составные имена нужно использовать в инструкциях ввода/вывода:

readln (m.number, m.marka, m.fio, m.address) ;

write (m.number:4, m.marka:10, m.fio:13, m. address: 23) ;

*Н*ельзя использовать в инструкциях ввода/вывода запись целиком (как и в случае массива). Например: **writein(m)** – ошибочная инструкция.

Допускается применение оператора присваивания к записям в целом, если они имеют один и тот же тип:

v:=m;

Обращение к полям записи имеет несколько громоздкий вид. Значительно упрощает написание использование имеющегося в языке Pascal оператора **with:**

with ПеременнаяТипаЗапись do Оператор;

Один раз указав имя переменной типа запись в операторе **with**, можно работать с именами полей, как с обычными переменными.

Например:

```
with m do begin
number:=1964;
marka:='Audi - 100';
fio:='Федорова Н.В.';
address:= 'ул. Красина 53 к.1 - 73';
end;
```

В ряде задач удобно пользоваться массивами из записей. Их можно описать, например, следующим образом:

```
type person = record
fio: string[20];
age: 1..99;
prof: string[30];
end;
var list : array[1..50] of person;
```

*Записи с вариантами

Возможности применения записей, имеющих строго определенную структуру, несколько ограничены. Однако в языке Pascal можно описать тип записи, содержащий несколько вариантов структуры. Записи этого типа называются *записями с вариантами*, они являются средством объединения похожих, но не идентичных по форме записей.

Запись с вариантами состоит из *фиксированной* и *вариантной* частей. Фиксированная часть задается так же, как и при описании простых записей. Вариантная часть формируется с помощью оператора **case ... of** и может состоять из нескольких вариантов. Оператор **case** задает особое поле записи — поле *признака*, которое определяет, какой из вариантов в данный момент будет активизирован. Значением признака в каждый текущий момент выполнения программы должна быть одна из расположенных в операторе **case** констант. Константа, служащая признаком, задает вариант записи и называется *константой выбора*.

В любой записи может быть только одна вариантная часть, и, если она есть, она должна располагаться за всеми фиксированными полями. Имена полей должны быть уникальными в пределах той записи, где они объявлены. Однако если записи содержат поля-записи, т. е. вложены одна в другую, имена могут повторяться на разных уровнях вложенности. Формат записи с вариантами таков:

```
type
ИмяТипа=record
 фиксированная часть
case ПолеПризнака : ИмяТипа of
КонстантаВыбора1 :     (Поле : Тип,...);
КонстантаВыбораN :    (Поле : Тип,...);
end;
```

Компоненты каждого варианта (идентификаторы полей и их типы) заключаются в круглые скобки. У части **case** нет отдельного **end,** как этого следовало бы ожидать по аналогии с обычным оператором **case.** Одно слово **end** заканчивает всю конструкцию записи с вариантами. Необходимо отметить, что количество полей каждого из вариантов не ограничено.

При использовании записей с вариантами необходимо придерживаться следующих правил:

все имена полей должны отличаться друг от друга, по крайней мере, одним символом, даже если они встречаются в разных вариантах;

запись может иметь только одну вариантную часть, причем вариантная часть должна размещаться в конце записи;

если вариантная часть, соответствующая какой-либо константе выбора, является пустой, то она записывается следующим образом:

КонстантаВыбораN();

*Домашнее задание

1. Составить опорный конспект лекции по теме «Символьные переменные и строки. Организация, размещение в памяти, процедуры и функции обработки строк» на основе презентации.

2. Программирование на языке Pascal. Рапаков Г. Г., Ржеуцкая С. Ю. СПб.: БХВ-Петербург, 2004, стр.256-268.

Написать программы:

- С клавиатуры вводится слово. Замените в нем все буквы «о» на «а» и наоборот.
- Напишите программу обращения слова. (например «корова» — «аворок»).