

# Тема 5.

# Строки. Множества

# Содержание

---

1. Символьные и строковые константы
2. Стандартные функции и процедуры обработки строк
3. Множества
4. Примеры решения задач

# 1. Символы и строки

---

В Теме 3 рассматривались произвольные массивы.  
Перейдем теперь к изучению массивов специального вида -  
линейных массивов, состоящих только из *символов*, -  
***строк***.

# Чем плох массив символов?

---

Это массив СИМВОЛОВ:

```
var B: array[1..N] of char;
```

- каждый символ – отдельный объект;
- массив имеет длину N, которая задана при объявлении

## Что нужно:

- обрабатывать последовательность СИМВОЛОВ как единое целое
- строка должна иметь переменную длину

# Описание символьных строк

---

В разделе *var строки* описываются следующим образом:

```
var <имя_строки>: string[<длина>];
```

**Максимальная длина строки - 255 символов.**

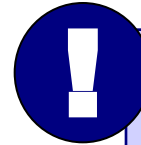
Нумеруются ее компоненты начиная с 0, но этот нулевой байт хранит *длину строки*.

Если *<длина>* не указана, то считается, что в *строке* 255 *символов*.

Поэтому для экономии памяти следует по возможности точно указывать длину используемых строк.

# Описание символьных строк

```
var s: string;
```



В *Delphi* это ограничение снято!

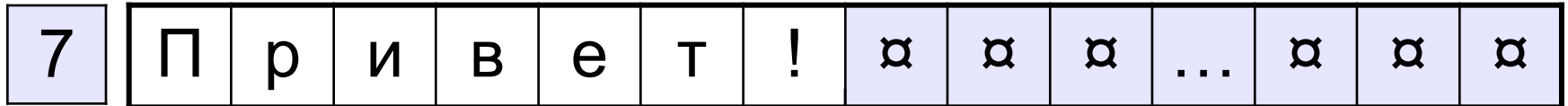
длина строки

s[3]

s[4]

1

↓  
255



s[1]

s[2]

1

20

```
var s: string[20];
```



Длина строки:

```
n := length ( s );
```

```
var n: integer;
```

# Описание символьных строк

---

Необходимо отметить, что один *символ* и *строка* длиной в один символ

```
var c: char;          s: string[1];
```

совершенно не эквивалентны друг другу.

Вне зависимости от своей реальной длины, строка относится к конструируемым структурированным типам данных, а не к базовым порядковым (см. Тему 2).



# Символ-константа и строка-константа: неименованные константы

---

В тексте программы на языке Паскаль последовательность любых символов, заключенная в *апострофы*, воспринимается как *символ* или *строка*.

Например:

```
c := 'z';      {c: char}  
s := 'abc';   {s: string}
```

# Символ-константа и строка-константа: неименованные константы

Константе автоматически присваивается "минимальный" тип данных, достаточный для ее представления:

*char* или *string[k]*.

Поэтому попытка написать

```
c := 'zzz';           {c: char}
```

вызовет ошибку уже на этапе компиляции.



Кроме того, если константа длиннее той переменной-строки, куда ваша программа пытается ее записать, то в момент присваивания произойдет усечение ее до нужной длины.

# Символ-константа и строка-константа: неименованные константы

---

*Пустая строка* задается двумя последовательными апострофами:

```
st := ' ';      { пустая строка }
```

Если же необходимо сделать так, чтобы среди символов строки содержался и сам *апостроф*, его нужно удвоить:

```
s := 'Don ' 't worry about the apostrophe! ' ;
```

Если теперь вывести на экран эту строку, то получится следующее:

Don't worry about the apostrophe!

# Символ-константа и строка-константа: нетипизированные константы

---

Все правила задания символов и строк как неименованных констант остаются в силе и при задании именованных нетипизированных констант в специальном разделе ***const***.

Например:

```
const c3 = ''';    {это один символ - апостроф!}  
s3 = 'Это строка';
```

# Символ-константа и строка-константа: типизированные константы

---

Типизированная константа, которая будет иметь тип *char* или *string*, задается в разделе *const* следующим образом:

```
const c4: char = ' ';  
        {это один символ - апостроф!}  
s4: string[20] = 'Это строка';
```

# Действия с символами: операции

---

Результатом унарной операции

**#<положительная неименованная константа  
целого типа>**

является *символ*, номер которого в таблице ASCII  
соответствует заданному числу.

Например,

```
#100    = 'd'  
#39     = ' '' '   {апостроф}  
#232    = 'ш'  
#1000   = 'ш'     {потому что (1000 mod 256) = 232}
```

# Действия с символами: операции

---

Кроме того, к символьным переменным, как и к значениям всех порядковых типов данных, применимы операции сравнения

<, <>, >, =,

результат которых также опирается на номера символов из таблицы ASCII.

# Действия с символами: стандартные функции

---

Функция

*chr(k:byte) : char*

"превращает"; номер символа в символ.

Действие этой функции аналогично действию операции #.

Например:

```
c := chr(48);    { c: char }  
                { c = '0' }
```



# Действия с символами: стандартные функции

---

Обратной к функции *chr()* является уже изученная нами функция *ord()*.

Таким образом, для любого числа *k* и для любого символа *c*

$$\begin{aligned} \mathit{ord}(\mathit{chr}(k)) &= k \quad ; \\ \mathit{chr}(\mathit{ord}(c)) &= c; \end{aligned}$$

# Действия с символами: стандартные функции

---

Стандартные процедуры и функции

*pred()*, *succ()*, *inc()* и *dec()*,

определенные для значений любого порядкового типа,

применимы также и к символам (значениям порядкового типа данных *char*).

Например:

```
pred('[') = 'z'  
succ('z') = '{'  
inc('a') = 'b'  
inc('c', 2) = 'e'  
dec('z') = 'y'  
dec(#0, 4) = '№' {#252}
```

# Действия с символами: стандартные функции

---

Стандартная функция

***uppercase(c: char):char***

превращает строчную букву в прописную.

Символы, не являющиеся строчными латинскими буквами, остаются без изменения (к сожалению, в их число попадают и все русские буквы).

# Ввод-вывод строковых переменных

---

Строки могут быть элементами *списка ввода–вывода*, при этом записывается имя строки без индекса.

Для ввода нескольких строковых данных следует пользоваться оператором ***Readln***, так как оператор ***Read*** в этих случаях может вести себя непредсказуемо.

- При вводе строковых переменных количество вводимых символов может быть меньше, чем длина строки. В этом случае вводимые символы размещаются с начала строки, а оставшиеся байты заполняются пробелами.
- Если количество вводимых символов превышает длину строки, лишние символы отбрасываются.

# Обращение к строковым переменным

---

Особенностью строковых переменных является то, что к ним можно обращаться

- как к скалярным переменным,
- так и к массивам.

Во втором случае применяется конструкция "переменная с индексом", что обеспечивает доступ к отдельным символам строки. При этом нижняя граница индекса равна 1.

Отдельный символ строки совместим с типом *char*.

Например,

```
S := St[20]; { обращение к 20 эл-ту строки St }  
Po := 'Компьютер'; { инициализация строки }  
Fillchar (a, sizeof(a), '0');  
{ заполнение строки a символами '0' }
```

# Обращение к строковым переменным

---

Например, если в программе определены

*Var S: string; C: char;*

и задано

*S:='Москва',*

то

*S[1]='М', S[2]='о'* и т. д.

# Обращение к строковым переменным

---

Элементы массива, составляющие строку можно переставлять местами и получать новые слова.

Пример 1. Вывод исходного слова справа налево: "авксом"

```
for i:=1 to N div 2 do begin  
    C:=S[i]; S[i]:=S[N-i+1]; S[N-i+1]:=C  
    end;  
Writeln(S);
```

Пример 2. Поиск и замена заданного символа в строке

```
for i:=1 to N do  
    if S[i]='_' then writeln('найден символ  
        пробел');  
for i:=1 to N do  
    if S[i]='/' then S[i]:='\';  
        { замена символа "/" на "\" }
```

## **2. Операции, стандартные функции и процедуры, выполняемые над строковыми переменными**



# Операции, выполняемые над строками

---

Для строк определены *операции*:

- присваивания,
- слияния (конкатенации, объединения),
- сравнения.

# Операция конкатенации

---

Результатом выполнения операции **конкатенации** "+", является строка, в которой исходные строки-операнды соединены в порядке их следования в выражении.

Например,

```
X := 'Пример';   Y := 'сложения';   Z := 'строк';  
writeln(X + Y + Z);  
writeln(Y + ' ' + Z + ' ' + X);
```

На экран будут выведены строки:

**Примерсложениястрок  
сложения строк Пример**

**'Кро'+ 'код'+ 'ил'**

позволит получить новую строку

**'Крокодил'**

# Операция конкатенации

---

Тип *String* допускает и *пустую строку* – строку, не содержащую символов:

*EmptyStr* := ' '; {подряд идущие кавычки}.

Она играет роль нуля (нейтрального элемента) операции конкатенации:

$$\mathit{EmptyStr} + X = X + \mathit{EmptyStr} = X.$$

# Операция сравнения

---

Строки - это единственный структурированный тип данных, для элементов которого определен порядок и, следовательно, возможны операции сравнения.

Сравнение строк происходит посимвольно, начиная с первого символа.

**Строки равны, если имеют одинаковую длину и посимвольно эквивалентны.**

Над строками определены также отношения (операции логического типа):

**=, <>, <, >, <=, >=.**

# Операция сравнения

---

Таким образом, каждый из строковых типов упорядочен лексикографически. Это означает, что

- порядок на строках согласован с порядком, заданным на символьном типе (*Char*);
- сравнение двух строк осуществляется посимвольно, начиная с первых символов;
- если строка *A* есть начало строки *B*, то  $A < B$ ;
- пустая строка – наименьший элемент типа.

Итак,

```
'abc' < 'xyz'  
'Иванов' < 'Иванова'  
'1200' < '45'  
'Татьяна' < 'татьяна'
```

# Функция *Length*

---

Формат:

```
Length (X : string) : byte;
```

Возвращает длину строки - аргумента ***X***. Причем, длина пустой строки ***Length(EmptyStr) = 0***.

Тип результата – ***Byte***.

## Примеры:

Исходные данные: ***S := 'крокодил';***

Оператор: ***j := length(S);***

Результат: ***j = 8***

Исходные данные: ***T := 'Компьютерный класс';***

Оператор: ***j := length(T);***

Результат: ***j = 18***

# Функция *Length*

**Задача:** ввести строку с клавиатуры и заменить все буквы «а» на буквы «б».

```
program qq;  
var s: string;  
    i: integer;  
begin  
    writeln('Введите строку');  
    readln(s);  
    for i:=1 to Length(s) do  
        if s[i] = 'a' then s[i] := 'б';  
    writeln(s);  
end.
```

ВВОД строки

длина строки

ВЫВОД строки

# Функция *Copy*

---

Формат:

```
Copy(X :string; Index, Count :byte): string;
```

Копирует (выделяет) подстроку строки *X*, начиная с позиции *Index* и содержащую следующие *Count* символов.

Тип результата – *String*.

Если *Index* больше длины строки, то результатом будет пустая строка.

Если же *Count* больше, чем длина оставшейся части строки, то результатом будет только ее "хвост":



# Функция *Сору*

```
s := '123456789';
```

с 3-его символа

6 штук

```
s1 := Сору ( s, 3, 6 );
```

'345678'

```
s2 := Сору ( s1, 2, 3 );
```

'456'

Исходные данные: ***S := 'крокодил';***

Оператор: ***b := сору(S, 2, 3);***

Результат: ***b = 'рок'.***

Исходные данные: ***T := 'Компьютерный класс***

***;***

Оператор: ***c := сору(T, 14, 53);***

Результат: ***c = 'класс'.***

***сору('abc3de Xyz', 2, 4) = 'bc3d'***

***сору('abc3de Xyz', 12, 4) = ''***

***сору('abc3de Xyz', 8, 14) = 'Xyz'***

# Функция *Concat*

---

Формат:

```
Concat (X1, X2, ..., Xk :string) :string
```

Объединение (конкатенация) строк или символов *X1*, *X2*, ..., *Xk* в указанном порядке. Другая форма записи: *X1+X2+ .. +Xk*.

Тип результата – *String*.

Если длина итоговой строки больше 255-ти символов, то произойдет отсечение "хвоста".

Кроме того, даже если результат конкатенации не был усечен, но программа пытается сохранить его в переменную заведомо меньшей длины, то усечение все равно состоится

# Функция *Concat*

---

Примеры:

Исходные данные:  $a := \text{'код'}$ ;  $b := \text{'ил'}$ ;

Оператор:  $S := \text{concat}(\text{'кро'}, a, b)$ .

Результат:  $S = \text{'крокодил'}$ .

$\text{concat}(\text{'abc'}, \text{'3de'}, \text{' '}, \text{'X'}, \text{'yz'}) = \text{'abc3de Xyz'}$

# Функция *Pos*

---

Формат:

```
Pos(Y, X : string) : byte;
```

Отыскивает первое вхождение строки *Y* в строке *X* (считая слева направо) и возвращает номер начальной позиции вхождения.

Если *X* не содержит *Y*, функция вернет 0 ( $\mathit{Pos}(Y, X) = 0$ ).

Тип результата – *Byte*.

Примеры:

Исходные данные:     *S* := 'крокодил';

Оператор:            *i* := *pos*('око', *S*);

Результат:           *i* = 3.

Оператор:            *i* := *pos*('я', 'крокодил');

Результат:           *i* = 0.

# Поиск в строке

Поиск в строке:

s[3]

```
var n: integer;
```

```
s := 'Здесь был Вася.' ;
n := Pos ( 'e' , s ) ;
if n > 0 then
    writeln('Буква e - это s[' , n , ']')
else writeln('Не нашли') ;
n := Pos ( 'Вася' , s ) ;
s1 := Copy ( s , n , 4 ) ;
```

3

n = 11

## Особенности:

- функция возвращает номер символа, с которого начинается образец в строке
- если слова нет, возвращается 0
- поиск с начала (находится **первое** слово)

# Процедура *Delete*

---

Формат:

```
Delete (X :string; Index, Count :byte);
```

Удаляет из строки *X* подстроку, начиная с позиции, заданной числом *Index*, длиной, заданной числом *Count*.

Тип результата – *String*.

Если число *Index* больше размера строки, то подстрока не удаляется.

Если число *Count* больше имевшегося количества, то удаляются символы до конца строки.

Тип результата – *String*.

# Процедура *Delete*

```
s := '123456789';
Delete ( s, 3, 6 );
```

6 штук

'12~~345678~~9'

'129'

строка  
меняется!

с 3-его символа

Исходные данные: ***S := 'крокодил';***

Оператор: ***delete(S, 4, 3);***

Результат: ***S = 'кроил'.***

Оператор: ***delete(S, 1, 1);***

Результат: ***S = 'роил'.***

Исходные данные: ***s = 'abc3de Xyz'***

Оператор: ***delete(S, 8, 13);***

Результат: ***S = 'abc3de '.***

# Процедура *Insert*

---

Формат:

```
Insert (Y, X :string; Index :byte);
```

Вставляет строку *Y* в строку *X*, начиная с позиции, заданной числом *Index*.

Тип результата – *String*.

Если *Index* выходит за конец строки, то подстрока *Y* припишется в конец строки *X*.

Если результат длиннее, чем допускается для строки *X*, произойдет его усечение справа.



# Процедура *Insert*

```
s := '123456789';
Insert ( 'ABC', s, 3 );
```

что  
вставляем

куда  
вставляем

начиная с 3-его символа

'12ABC3456789'

```
Insert ( 'Q', s, 5 );
```

'12ABQC3456789'

Исходные данные: ***S := 'крокодил';***

Оператор: ***d := copy(S, 3, 3);***

Результат: ***d = 'око'.***

Оператор: ***insert('H', d, 3);***

Результат: ***d = 'окHo'.***

# Примеры

```

s := 'Вася Петя Митя';
n := Pos ( 'Петя', s );
Delete ( s, n, 4 );
Insert ( 'Лена', s, n );

```

6  
'Вася Митя'  
'Вася **Лена** Митя'

```

s := 'Вася Петя Митя';
n := length ( s );
s1 := Copy ( s, 1, 4 );
s2 := Copy ( s, 11, 4 );
s3 := Copy ( s, 6, 4 );
s := s3 + s1 + s2;
n := length ( s );

```

14  
'Вася'  
'Митя'  
'Петя'  
'ПетяВасяМитя'  
12

# Процедура *Val*

---

Формат:

```
VAL(St :string; Ibr :<арифметический_тип>;  
      Cod :byte );
```

Преобразует строку символов ***St*** в величину целочисленного или вещественного типа и помещает результат в ***Ibr***.

***Ibr*** является внутренним представлением числа, записанного в символьном формате.

Значение ***St*** не должно содержать незначащих пробелов в начале и в конце.

***Cod*** – целочисленная переменная, указывающая номер позиции первого неправильного символа, или равная 0 в случае успешного преобразования.

# Процедура *Val*

Преобразование из строки в число:

```
var N: integer;
    X: real;
    s: string;
```

```
s := '123';
Val ( s, N, r ); { N = 123 }
    { r = 0, если ошибки не было
      r – номер ошибочного символа }
s := '123.456';
Val ( s, X, r ); { X = 123.456 }
```

Исходные данные: ***St := '1500';***

Оператор: ***Val(St, lbr, Cod);***

Результат: ***lbr = 1500 Cod = 0***

Исходные данные: ***St := '4.8A+03';***

Оператор: ***Val(St, lbr, Cod);***

Результат: ***lbr = ??? Cod = 5***

# Процедура *Str*

---

Формат:

```
STR (Ibr [:M [:N] ], St:string );
```

Преобразует числовое значение величины *ibr* в строковое и помещает результат в строку *St*.

*M* задает общее количество символов, получаемых в строке,

*N* - для вещественных чисел (типа *real*) задает количество цифр в дробной части.

Тип результата – *String*.

Если в формате указано недостаточное для вывода количество разрядов, поле вывода расширяется автоматически до нужной длины.

Если число короче указанных величин, то спереди и/или сзади оно будет дополнено пробелами.

# Процедура *Str*

## Преобразование из числа в строку:

```
var N: integer;
    X: real;
    s: string;
```

```
N := 123;
Str ( N, s );           { '123' }
X := 123.456;
Str ( X, s );           { '1.234560E+002' }
Str ( X:10:4, s );     { ' 123.456 ' }
```

Исходные данные: ***lbr := 1500;***

Оператор: ***str(lbr:6, S);***

Результат: ***S = ' \_\_1500'.***

Исходные данные: ***lbr := 4.8E+03;***

Оператор: ***str(lbr:10, S);***

Результат: ***S = ' \_\_\_\_\_4800'.***

# **3. Примеры обработки СИМВОЛЬНЫХ ДАННЫХ**

**Пример 1.** Проверить, является ли заданный символ  $S$  строчной гласной буквой русского алфавита.

**Решение.**  $pos(S, 'аэоуыяеёи') > 0$

**Пример 2.** Выделить часть строки после первого пробела.

**Решение.**

$copy(s, pos(' ', s)+1, Length(s)-pos(' ', s))$

**Пример 3.** Удалить последний символ строки  $S$ .

**Решение.**  $delete(S, Length(S), 1)$

**Пример 4.** Выделить из строки  $S$  подстроку между  $i$ -й и  $j$ -й позициями, включая эти позиции.

**Решение.**  $copy(s, i, j-i+1)$



**Пример 5.** Подсчитать, сколько раз в заданной строке встречается указанная буква.

**Решение 1.** Обозначим заданную строку - **s**, а искомую букву - **a**. Тогда для решения задачи будем просматривать заданную строку посимвольно и каждый символ сравнивать с заданной буквой.

```
k:=0; { количество указанных букв в строке }
for i:=1 to Length(s) do
  if copy(s, i, 1)=a then k:=k+1;
```

**Решение 2.** Ищем положение указанной буквы в строке до тех пор, пока ее удастся найти. Затем отбрасываем ту часть строки, где была найдена указанная буква, и повторяем поиск.

```
k:=0; j:=pos(a, s); { позиция 1-го вхождения a в строку s }
while j<>0 do begin
  k:=k+1;
  s:=copy(s, j+1, Length(s)-j);
  j:=pos(a, s)
end;
```

**Пример 6.** Назовем словом любую последовательность букв и цифр. Строка состоит из слов, разделенных одним или несколькими пробелами. Удалить лишние пробелы, оставив между словами по одному пробелу.

**Решение.** Лишними пробелами называются второй, третий и т.д., следующие за первым пробелом. Следовательно, чтобы найти лишний пробел, нужно искать два пробела, стоящие рядом, и удалять второй пробел в каждой найденной паре.

```
j:=pos(' ', s);  
while j<>0 do begin  
    delete(s, j, 1);  
    j:=pos(' ', s)  
end;
```

# Пример решения задачи

---

**Задача:** Ввести имя, отчество и фамилию. Преобразовать их к формату «фамилия-инициалы».

**Пример:**

Введите имя, фамилию и отчество:

**Василий Алибабаевич Хрюндиков**

Результат:

**Хрюндиков В.А.**

**Алгоритм:**

- найти первый пробел и выделить имя
- удалить имя с пробелом из основной строки
- найти первый пробел и выделить отчество
- удалить отчество с пробелом из основной строки
- «сцепить» фамилию, первые буквы имени и фамилии, точки, пробелы...

# Программа

---

```
program qq;
var s, name, otch: string;
    n: integer;
begin
    writeln('Введите имя, отчество и фамилию');
    readln(s);
    n := Pos(' ', s);
    name := Copy(s, 1, n-1); { вырезать имя }
    Delete(s, 1, n);
    n := Pos(' ', s);
    otch := Copy(s, 1, n-1); { вырезать отчество }
    Delete(s, 1, n);          { осталась фамилия }
    s := s + ' ' + name[1] + '.' + otch[1] + '.';
    writeln(s);
end.
```

# Задания

---

Ввести имя файла (возможно, без расширения) и изменить его расширение на «.exe».

## Пример:

Введите имя файла:

qqq

Результат:

qqq.exe

Введите имя файла:

qqq.com

Результат:

qqq.exe

Ввести полный путь к файлу и «разобрать» его, выводя каждую вложенную папку с новой строки

## Пример:

Введите путь к файлу:

C:\Мои документы\09ИТ-3\Вася\qq.exe

Результат:

C:

Мои документы

09ИТ-3

Вася

qq.exe

# Посимвольный ввод

---

**Задача:** с клавиатуры вводится число  $N$ , обозначающее количество футболистов команды «Шайба», а затем –  $N$  строк, в каждой из которых – информация об одном футболисте таком формате:

*<Фамилия> <Имя> <год рождения> <голы>*

Все данные разделяются одним пробелом. Нужно подсчитать, сколько футболистов, родившихся в период с 1988 по 1990 год, не забили мячей вообще.

## Алгоритм:

```
for i:=1 to N do begin
  { пропускаем фамилию и имя }
  { читаем год рождения Year и число голов Gol }
  if (1988 <= Year) and (Year <=1990) and
     (Gol = 0) then { увеличиваем счетчик }
end;
```

# ПОСИМВОЛЬНЫЙ ВВОД

## Пропуск фамилии:

```
var c: char;
```

```
repeat  
  read(c);  
until c = ' '; { пока не встретим пробел }
```

## Пропуск имени:

```
repeat read(c); until c = ' ';
```

## Ввод года рождения:

```
var Year: integer;
```

```
read(Year); { из той же введенной строки }
```

## Ввод числа голов и переход к следующей строке:

```
readln(Gol); { читать все до конца строки }
```

```
var Gol: integer;
```

# Программа

```
program qq;
var c: char;
    i, N, count, Year, Gol: integer;
begin
  writeln('Количество футболистов');
  readln(N);
  count := 0;
  for i:=1 to N do begin
    repeat read(c); until c = ' ';
    repeat read(c); until c = ' ';
    read(Year);
    readln(Gol);
    if (1988 <= Year) and (Year <= 1990) and
        (Gol = 0) then count := count + 1;
  end;
  writeln(count);
end.
```



# ПОСИМВОЛЬНЫЙ ВВОД

Если фамилия нужна:

```
var fam: string;
```

```
fam := ''; { пустая строка }
repeat
  read(c); { прочитать символ }
  fam := fam + c; { прицепить к фамилии }
until c = ' ';
```

Вместо read(Year):

```
var s: string;
```

```
s := ''; { пустая строка }
repeat
  read(c); { прочитать символ }
  s := s + c; { прицепить к фамилии }
until c = ' ';
```

```
Val(s, Year, r); { строку - в число }
```

# ПОСИМВОЛЬНЫЙ ВВОД

Если нужно хранить все фамилии:

```
const MAX = 100;  
var fam: array[1..MAX] of string;  
...  
fam[i] := ''; { пустая строка }  
repeat  
    read(c); { прочитать символ }  
    fam[i] := fam[i] + c;  
until c = ' ';
```

**МАССИВ  
СИМВОЛЬНЫХ  
СТРОК**

# 4. Множества

---

**Множество** – это структурированный тип данных, представляющий собой набор взаимосвязанных по какому-либо признаку или группе признаков объектов, которые можно рассматривать как единое целое.

Каждый объект в множестве называется **элементом множества**.

Все элементы множества должны принадлежать одному из скалярных типов, кроме вещественного. Этот тип называется **базовым типом** множества.

Базовый тип задается диапазоном или перечислением.

---

**Область значений типа множество** – набор всевозможных подмножеств, составленных из элементов базового типа.

В выражениях на Паскале значения элементов множества указываются в квадратных скобках:

***[1, 2, 3, 4], ['a', 'b', 'c'], ['a'..'z']***.

Если множество не имеет элементов, оно называется ***пустым*** и обозначается как ***[]***.

---

Количество элементов называется его **мощностью**.

Количество элементов множества не должно превышать 256, соответственно номера значений базового типа должны находиться в диапазоне 0..255.

Важное отличие множества от остальных структурированных типов состоит в том, что его элементы не являются упорядоченными.

# Описание множества

---

Формат записи множественного типа и переменной, относящейся к нему:

```
Type <имя типа> = set of  
      <тип_элементов_множества>;  
Var <идентификатор, ...> : <имя типа>;
```

В разделе **var** множества описываются следующим образом (без предварительного описания типа):

```
Var <имя_множества>: set of  
      <тип_элементов_множества>;
```

Элементы могут принадлежать к любому порядковому типу, размер которого не превышает 1 байт (256 элементов).

# Описание множества: примеры

*type*

*Simply = set of 'a'..'h',*

*Number = set of 1..31;*

*Var Pr : Simply;*

*N : Number;*

*Letter : set of char;*

*Pr* может принимать значения символов латинского алфавита от 'a' до 'h';

*N* – любое значение в диапазоне 1...31;

*Letter* – любой символ;

*Var s2: set of 'a'..'z', 'A'..'Z';*

{множество из 52-х элементов}

*s3: set of 0..10;* {множество из 11-ти элементов}

*s4: set of boolean;* {множество из 2-х элементов}



# Множество-константа: неименованная константа

---

Множество можно задать неименованной константой прямо в тексте программы. Для этого необходимо заключить список элементов создаваемого множества в квадратные скобки:

**[<список\_элементов>]**

*Список элементов* может быть задан перечислением элементов нового множества через запятую, интервалом или объединением этих двух способов.

Элементы и границы интервалов могут быть переменными, константами и выражениями. Если левая граница интервала окажется больше правой, результатом будет пустое множество.

# Множество-константа: неименованная константа

---

Примеры конструирования и использования различных множеств:

```
if c in ['a', 'e', 'i', 'o', 'u'] then  
    writeln('Гласная буква');  
if set1 < [k*2+1 .. n, 13] then  
    set1 := [];
```

# Множество-константа: нетипизированная константа

---

*Множество* - это структурированный тип данных, поэтому его невозможно задать нетипизированной константой!

# Множество-константа: типизированная константа

Задать множество как типизированную константу можно в разделе **const**:

```
<ИМЯ_КОНСТАНТЫ> : set of  
  <ТИП_ЭЛЕМЕНТОВ> = [<СПИСОК_ЭЛЕМЕНТОВ>];
```

Например,

```
type    cipher = set of '0'..'9';  
const odds: cipher = ['1', '3', '5', '7', '9'];  
       vowels: set of  
'a'..'z' = ['a', 'o', 'e', 'u', 'i'];
```

# 5. Операции с множествами

---

Использование в программе данных типа **set** дает ряд преимуществ:

- значительно упрощаются сложные операторы **if**,
- увеличивается степень наглядности программы и понимания решения задачи,
- экономится память, время компиляции и выполнения.

Имеются и отрицательные моменты.

Основной из них – отсутствие в языке Паскаль средств ввода-вывода элементов множества, поэтому программист сам должен писать соответствующие процедуры.

---

При работе с множествами допускается использование операций:

- отношения  $=$ ,  $\langle \rangle$ ,  $\geq$ ,  $\leq$
- операции ***IN***
- объединения множеств
- пересечения множеств
- разности множеств.

Результатом выражений с применением первых двух операций является значение ***True*** или ***False***.

# Операция «Равно» (=)

Два множества **A** и **B** считаются равными, если они состоят из одних и тех же элементов.

Порядок следования элементов в сравниваемых множествах значения не имеет.

Например,

Значение <b>A</b>	Значение <b>B</b>	Выражение	Результат
<b>[1, 2, 3, 4]</b>	<b>[1, 2, 3, 4]</b>	<b>A=B</b>	<b>True</b>
<b>['a', 'b', 'c']</b>	<b>['c', 'a']</b>	<b>A=B</b>	<b>False</b>
<b>['a'..'z']</b>	<b>['z'..'a']</b>	<b>A=B</b>	<b>True</b>



# Операция «Неравно» ( $\langle \rangle$ )

Два множества **A** и **B** считаются не равными, если они отличаются по мощности или по значению хотя бы одного элемента.

Например,

Значение <b>A</b>	Значение <b>B</b>	Выражение	Результат
<b><i>[1, 2, 3]</i></b>	<b><i>[3, 1, 2, 4]</i></b>	<b><i>A&lt;&gt;B</i></b>	<b><i>True</i></b>
<b><i>['a'..'z']</i></b>	<b><i>['b'..'z']</i></b>	<b><i>A&lt;&gt;B</i></b>	<b><i>True</i></b>
<b><i>['c'..'t']</i></b>	<b><i>['t'..'c']</i></b>	<b><i>A&lt;&gt;B</i></b>	<b><i>False</i></b>

# Операция «Больше или равно» ( $\geq$ )

Операция «больше или равно» используется для определения принадлежности множеств.

Результат операции  $A \geq B$  равен *True*, если все элементы множества  $B$  содержатся в множестве  $A$ . В противном случае результат равен *False*.

$$(A \supseteq B)$$

Например,

Значение $A$	Значение $B$	Выражение	Результат
$[1, 2, 3, 4]$	$[2, 3, 4]$	$A \geq B$	<i>True</i>
$['a'..'z']$	$['b'..'t']$	$A \geq B$	<i>True</i>
$['z', 'x', 'c']$	$['c', 'x']$	$A \geq B$	<i>True</i>

# Операция «Меньше или равно» ( $\leq$ )

Эта операция используется аналогично предыдущей операции, но результат выражения  $A \leq B$  равен *True*, если все элементы множества  $A$  содержатся в множестве  $B$ . В противном случае результат равен *False*.

$$(A \subseteq B)$$

Например,

Значение $A$	Значение $B$	Выражение	Результат
$[1, 2, 3, 4]$	$[2, 3, 4]$	$A \leq B$	<i>False</i>
$['d'..'h']$	$['z'..'a']$	$A \leq B$	<i>True</i>
$['a', 'v']$	$['a', 'n', 'v']$	$A \leq B$	<i>True</i>

# Операция *IN*

Эта операция используется для проверки принадлежности какого-либо значения указанному множеству. Обычно применяется в условных операторах.

**( $a \in B$ )**

Например,

Значение	Выражение	Результат
<b><i>A='v'</i></b>	<b><i>If A in ['a'..'n'] then ...</i></b>	<b><i>False</i></b>
<b><i>A=X1</i></b>	<b><i>If A in [X0, X1, X2, X3] then ...</i></b>	<b><i>True</i></b>

При использовании операции *in* проверяемое на принадлежность значение и множество в квадратных скобках не обязательно предварительно описывать в разделе описаний.

# Операция *IN*

---

Операция *in* позволяет эффективно и наглядно производить сложные проверки условий, заменяя иногда десятки других операций.

Например, выражение

```
if (a=1) or (a=2) or (a=4)  
    or (a=5) or (a=6) then ...
```

можно заменить более коротким выражением

```
if a in [1..6] then ...
```

# Операция отрицания *IN*

---

Часто операцию *in* пытаются записать с отрицанием:

***X NOT in M***

***(x∉M)***

Такая запись является ошибочной, так как две операции следует подряд.

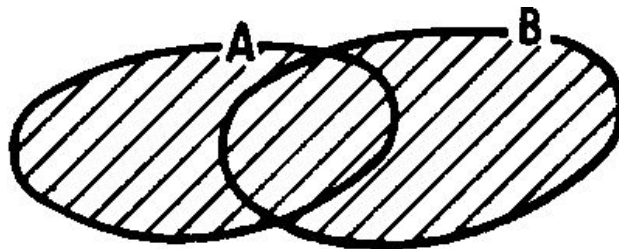
Правильная конструкция имеет вид

***NOT (X in M)***

# Операция «Объединение» (+)

Объединением двух множеств является третье множество, содержащее элементы обоих множеств ( $A \cup B$ ).

Графическая интерпретация:



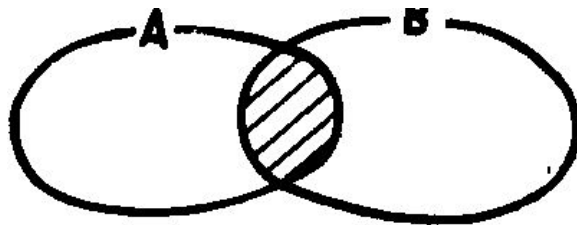
Например,

Значение $A$	Значение $B$	Выражение	Результат
$[1, 2, 3]$	$[1, 4, 5]$	$A+B$	$[1, 2, 3, 4, 5]$
$['A'..'D']$	$['E'..'Z']$	$A+B$	$['A'..'Z']$
$[]$	$[]$	$A+B$	$[]$

# Операция «Пересечение» (\*)

Пересечением двух множеств является третье множество, которое содержит элементы, входящие одновременно в оба множества ( $A \cap B$ ).

Графическая интерпретация:



Например,

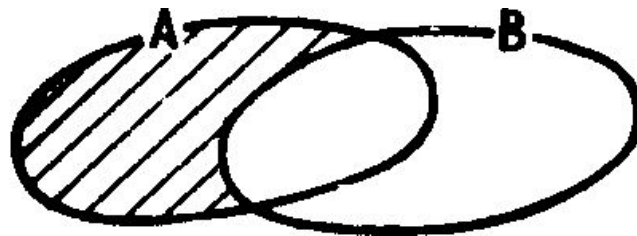
Значение $A$	Значение $B$	Выражение	Результат
$[1, 2, 3]$	$[1, 4, 2, 5]$	$A * B$	$[1, 2]$
$['A'..'Z']$	$['B'..'R']$	$A * B$	$['B'..'R']$
$[]$	$[]$	$A * B$	$[]$



# Операция «Разность» ( $-$ )

Разностью двух множеств является третье множество, которое содержит элементы первого множества, не входящие во второе множество ( $A \setminus B$ ).

Графическая интерпретация:



Например,

Значение $A$	Значение $B$	Выражение	Результат
$[1, 2, 3, 4]$	$[3, 4, 1]$	$A - B$	$[2]$
$['A'..'Z']$	$['D'..'Z']$	$A - B$	$['A'..'C']$
$[X1, X2, X3, X4]$	$[X4, X1]$	$A - B$	$[X2, X3]$

---

Не существует никакой процедуры, позволяющей распечатать содержимое множества. Это придется делать следующим образом:

```
{s: set of type1; k: type1}  
for k:= min_type1 to max_type1 do  
if k in s then  
write(k);
```

## **6. Примеры использования символов, строк и множеств**

# Пример 1

---

Задано множество целых положительных чисел от 1 до  $n$ .

Создать из элементов этого множества такие подмножества, элементы которых удовлетворяют следующим условиям:

- Элементы подмножества не больше 10;
- Элементы подмножества кратны 8;
- Элементы подмножества не кратны 3 и 6.

# Пример 1

```
Program mnoj;  
Const n=100;  
Var mn1, mn2, mn3: set of 1..n;  
    k: integer;  
Begin  
    {задание начальных значений подмножеств (пустые)}  
    mn1:=[]; mn2:=[]; mn3:=[];  
    for k:=1 to n do begin {создание подмножеств}  
        if k<=10 then mn1:=mn1 + [k];  
        if k mod 8 =0 then mn2:=mn2 + [k];  
        if (k mod 3<>0) and (k mod 6<>0) then  
            mn3:=mn3 + [k];  
    end;  
end;
```

# Пример 1

```
{печать полученных множеств}
writeln('подмножество чисел не больших 10');
for k:=1 to n do
    if k in mn1 then write(k:4);

writeln('подмножество чисел кратных 8');
for k:=1 to n do
    if k in mn2 then write(k:4);

writeln('подмножество чисел не кратных 3 и
6');
for k:=1 to n do
    if k in mn3 then write(k:4);

end.
```

# Пример 2

mn1 – содержит все  
встречающиеся буквы в строке

е бу  
сте т

mn2 – содержит  
парные буквы

рые

```

program mnogestvo;
Var mn1, mn2: set of char;
    i: integer;   Stroka: string;
Begin
  writeln('Введите строку ');  readln(Stroka);
  mn1:=[];  mn2:=[];           {Пустые множества}
  for i:=1 to Length(Stroka) do
    if Stroka[i] in mn1 then
      mn2:=mn2+[Stroka[i]]
    else  mn1:=mn1 + [Stroka[i]];
  for i:=1 to Length(Stroka) do
    if (not(Stroka[i] in mn2)) then
      writeln(Stroka[i]);
end.

```

Вывод тех букв, которых  
нет в множестве mn2

## Пример 3

Оставить в строке только первое вхождение каждого символа, взаимный порядок оставленных символов сохранить.

```
program z3;  
  var s: set of char;  
      inp, res: string; i: byte;  
begin  
  s:=[]; res:= '';  
  for i:=1 to length(inp) do  
    if not(inp[i] in s) then begin  
      res:= res+inp[i];  
      s:= s+[inp[i]];  
    end;  
end.
```

*s* – содержит все встречающиеся буквы в строке;  
*inp* – исходная строка;  
*res* – результирующая строка



## Пример 4

Оставить в строке только последнее вхождение каждого символа, взаимный порядок оставленных символов сохранить.

*inp* – исходная строка;  
*res* – результирующая строка

```
program z3;  
  var inp, res: string;  
      i: byte;  
begin  
  res := '';  
  for i:=1 to length(inp) do begin  
k := pos(inp[i], res);  
  if k <> 0 then delete(res, k, 1);  
res := res+inp[i];  
  end;  
end.
```

## Пример 5

---

Задано предложение, состоящее из слов, разделенных одним или несколькими пробелами. Определить самое длинное слово предложения.

*Решение.* Чтобы выделить окончание слова, нужно анализировать два символа:

- первый символ должен быть отличен от пробела,
- а второй должен быть пробелом.

Для одинаковой обработки всех символов предложения добавим к концу предложения дополнительный символ - пробел.

Как только обнаружится конец слова, вычислим его длину и проверим на максимум:

# Пример 5

```

smax := ' ';           { слово максимальной длины }
readln(s);           { исходное предложение }
s := s + ' ';       { исходное предложение с доп. пробелом }
ss := ' ';         { текущее слово предложения }
for i := 1 to length(s) - 1 do
  { просмотр предложения по два символа }
  if s[i] <> ' ' and s[i+1] = ' ' then begin
    { если текущий символ не пробел, а следующий - пробел }
    ss := ss + s[i];   { дописали последний символ }
    if length(smax) < length(ss) then smax := ss;
    { если длина нового слова больше, чем длина smax,
      то запоминаем его }
    ss := ' ';       { готовим место для следующего слова }
  end
  else if s[i] <> ' ' then ss := ss + s[i];
  { если текущий символ не пробел, то запоминаем его в слове }.

```

## Пример 5

---

Из данной символьной строки выбрать все цифры и сформировать другую строку из этих цифр, сохранив их последовательность.

```
Program Stroki;  
Var S1, S2:string;  
Begin  
Write('Введите строку');  
Readln(S1);  
S2:= '';  
For i:=1 to length(S1) do  
    If (S1[i]>='0') and (S1[i]<='9') then  
        S2:=S2 + S1[i];  
Writeln('Результат', S2)  
End.
```