17. Множества.

17.1 Множества в Паскале и в математике. Сходства и различия между ними.

Понятие множества в Паскаль пришло из математики. В математике множество - это такой объект, который представляет коллекцию других объектов, порядок расположения и тип которых не играет никакой роли. Различать будем понятия множества в математике и в Паскале таким образом:

1

В Паскале элементы множества должны принадлежать одному и тому же базовому типу.

Количество элементов: в математике *не ограничено*; в Паскале число элементов множества <= *256* (в памяти множество может занимать до 256:8 - 32 байт). При этом номера этих элементов могут быть от *0* до *255* (только положительными). Поэтому базовым типом множества на Паскале может быть только такой тип, число возможных значений которого укладывается в диапазон (*0-255*): символьный, перечисляемый, булевский, тип-диапазон, *byte*.

Порядок элементов не имеет значения ни в математике, ни в Паскале.

Так, на Паскале множество:

[1,2,3]
[3,2,1]
[2,1,3]

Это одно и то же множество

17.2 Объявление множества на Паскале

В общем виде объявление множества на Паскале имеет следующий вид:

```
var
s : set of базовый тип;
```

диапазон; *byte; char, boolean;* перечисление

S - переменная типа множества (или множественная переменная).

множественный тип

Указание базового типа автоматически определяет **совокупность возможных значений множественной переменной.** В эту совокупность входят все возможные неповторяющиеся наборы (комбинации) значений базового типа, в том числе и наборы из одного элемента, из двух, трех и т.д.,

включая пустое множество (оно автоматически входит в любое множество). Эти наборы и составляют возможные значения определенного множественного типа.

ПРИМЕР.

```
var
 s: set of 1..3;
```

[ ], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]

Возможные значения переменной S (реальным (текущим) значением будет то, которое будет присвоено явно)

Примечания:

1. Значение переменной-множества представляет собой один конкретный набор из множества возможных наборов неповторяющихся значений базового типа [1, 2, 3].
2. В каждый конкретный момент времени переменная-множество может принимать только один из возможных наборов значений.
3. Объявление переменной множественного типа не вызывает автоматического присваивания ей значения. Под нее только выделяется память.

Хранится переменная типа множества в памяти очень компактно - в виде массива битов (битовых строк). В этих битовых строках (массивах битов) хранятся не значения базового типа, а информация о наличии или отсутствии их (значений базового типа) в множестве: каждому отдельному значению базового типа соответствует отдельный разряд (бит). *Единица* в этом разряде соответствует ситуации наличия данного значения базового типа в данный момент в значении переменной типа множество. *Ноль* означает, что данное значение базового типа в данный момент в множестве отсутствует. Пустое множество хранится в виде нулевой (состоящей из нулей) битовой строки.

Благодаря такому представлению множеств (в виде битовых строк или кодов), операции с множествами выполняются в машине очень быстро – это достоинство.

При работе с множествами есть ***недостаток*:** значение переменных множественного типа нельзя вводить и выводить в процедурах (ввода - вывода). Тем не менее значение элемента множества можно наблюдать в окне отладчика.

17.3 Присваивание значений множествам. Конструктор множества

Присваивание значений множественной переменной производится с помощью т.н. конструктора множеств и происходит в исполняемой части программы.

```
Begin
 s := [1,3];
```

Конструктор множества

В общем случае конструктор множества представляет из себя заключенный в квадратные скобки список констант, переменных или выражений определенного типа - базового типа множества. Вместо отдельных констант в этом списке могут использоваться диапазоны.

ПРИМЕР.

```
S := [1, 2, 3, 4, 5];
s := [1..5];
var
        s: set of char;

begin
 s := ['a'..'z', 'A'..'Z'];
```

Одно и то же

Это сокращенная запись следующего списка букв латинского алфавита: 'a', 'b', 'c', 'd', 'e',.....'z', 'A' ,'B', 'C', 'D', 'E',…....,'Z'

S := ‘a’..’z’; -- неправильно (нет квадратных скобок)

17.4 Операции над множествами.

Над множествами возможны три операции. Все операции двухместные.

Операндами в этих операциях могут быть как переменные типа множеств, так и конструкторы множеств. Операнды должны принадлежать к одному и тому же базовому (множественному) типу.

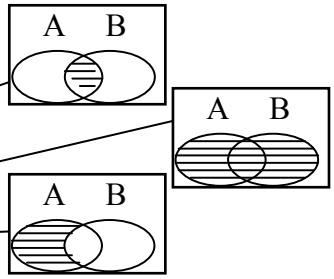
```
Var
 a, b: set of char;
 c: set of char;
```

a, b, c - множественные переменные. *char* - базовый тип. *Set of char* - множественный тип.

В данном случае *a* и *b* принадлежат к одному и тому же множественному типу, а *c* - к другому, хотя переменные имеют формально один и тот же тип.

Таблица операций над множествами.

Операция	Математика	Паскаль
Пересечение	$a \cap b$	*a * b*
Объединение	$A \cup b$	*a + b*
Разность	*a \ b*	*a - b*



Определение 1. ***Пересечение*** множеств - новое множество, состоящее из элементов, принадлежащих одновременно множествам *A* и *B.*

Определение 2. ***Объединение*** множеств - новое множество, в которое входят элементы или из элементов множества A или из элементов множества B или из элементов, принадлежащих тому и другому одновременно.

Определение 3. ***Разность*** множеств - новое множество, в которое входят элементы уменьшаемого множества (A), не входящие в число элементов вычитаемого множества (B).

Примечание. Если при выполнении операции *объединения (A + B)* включаемые в A из B элементы уже присутствуют в множестве A и, если при выполнении операции *разности (A - B)* вычитаемые из A элементы отсутствуют в множестве A, то сообщения об ошибке не будет. Операции просто не будут выполняться.

Последние две операции используются для выполнения следующих действий:

1). *(A+B)* используются для включения в множество отдельных элементов.

2). *(A-B)* используется для исключения отдельных элементов из множества.

```
Var
 s :set of ['a'..'z'];
begin
 s := [];          Пустое множество
 s := s+'a';       Ошибка: a не является ни конструктором, ни переменной множественного типа.
 s :=  s+['a']; //- верно, включает элемент 'a' в множество s ≡ include(s, 'a').
 s :=  s-['a']; //- верно, исключает элемент 'a' из множества s ≡ exclude(s, 'a').
```

Для ввода значения множества и вывода содержимого множества **нельзя использовать операторы *read* и *write*.**

ПРИМЕР. Рассмотрим программу, которая заполняет множество **поэлементно**.

Примечание. Пусть признаком завершения ввода является '.' (точка) во входном потоке:

1) Цикл с постусловием

```
var
  s : set of char; {множество}
  c : char;                {символ , вводимый с клавиатуры}
begin
  s := [];                 Подготовка цикла
  repeat
     read (c);
    s :=  s+[c]; //здесь используется текущее значение C
  until (c='.');  //здесь проверяется текущее значение C
  s := s - ['.'];
end.
```

2) Цикл с предусловием

```
 var
  s : set of char;         {множество}
```

```
c : char;                {символ , вводимый с клавиатуры}
begin
  s := [];
  c := #0;
while c <> '.' do
  begin
    read (c);
    s := s+[c];
  end;
end.
```

Подготовка цикла

Здесь проверяется предыдущее, а не текущее значение *c*

Вставка

надо вставить

```
s := s - ['.'];
```

Правильный вариант:

```
.......
read(c);
while c<> '.' do
 begin
   s := s + [c];
   read(c);
 end;
.......
```

В данной программе не хватает одной строчки. Последним действием в цикле будет включение в число символов множества разделителя (*'.'*). Эту точку*'.'* из множества надо исключить (смотри вставку).

ПРИМЕР. Сформировать ***множество четных чисел*** в диапазоне от *1* до *100*.

Схема решения:

1. Сначала надо сформировать множество чисел от *1* до *100*, которое включает все четные и нечетные числа.
2. Исключить все нечетные числа.

```
Const
 n = 100;
var
 s : set of 1..n; //формируемое множество
 k : 1..n;        //добавляемые в множество числа
begin
{начальное заполнение}
 s := [1..n];
 {исключение нечетных чисел}
 for k := 1 to n do
  if  odd(k) then
    s :=  s - [k];
end.
```

В множество *s* включается совокупность чисел от *1* до *100*

{*odd* - возвращает "истина", если число - нечетное}

17.5 Сравнение множеств.

Замечание. Можно сравнивать множества только совместимых базовых типов (как и в п.17.4).

	Математика	Паскаль
1	$A = B$	$A = B$
2	$A \neq B$	$A <> B$
3	$A \supseteq B$	$A >= B$
4	$A \subseteq B$	$A <= B$
5	$x \in A$	x *in* A

x должен быть базового типа множества

Все операции возвращают **логический результат**. Для каждой операции правила формулируются следующим образом:

1. *Равенство*: (*true*) только тогда, когда 2 множества имеют одинаковый набор символов.
2. *Неравенство*: (*true*) когда набор символов различный.
3. $A \supseteq B$ возвращает *true*, если все элементы множества B входят в множество A.
4. $A \subseteq B$ возвращает *true*, если все элементы множества A входят в множество B.
5. $x \in A$ возвращает *true*, если x - (элемент базового типа множества A) имеется в множестве A.

17.6 Применение множеств.

1-й Случай: Множества используют для того, чтобы *исключить большое количество последовательных проверок*.

ПРИМЕР. При вводе символа с клавиатуры надо определить, является ли текущий символ символом английского алфавита. Без использования множеств это можно сделать тремя путями:

1 путь.

```
. Var
   c: char;
  begin
   readln(c);
   if (c = 'a') or (c = 'b') or ... or (c = 'z') or (c = 'A') or ... or (c = 'Z')
    then <с - буква>...
```

52 проверки

3 путь:
case c of
'a'..'z',
A'..'Z': <с - буква>
……

2 путь

```
Var
 c: char;
begin
 readln(c);
 if ((c >= 'a') and (c <= 'z')) or ((c >= 'A') and (c <= 'Z'))
  then <с - буква>...
```

С использованием множеств те же самые действия можно записать следующим образом:

```
if c in ['a' .. 'z', 'A' .. 'Z'] then <с - буква>...
```

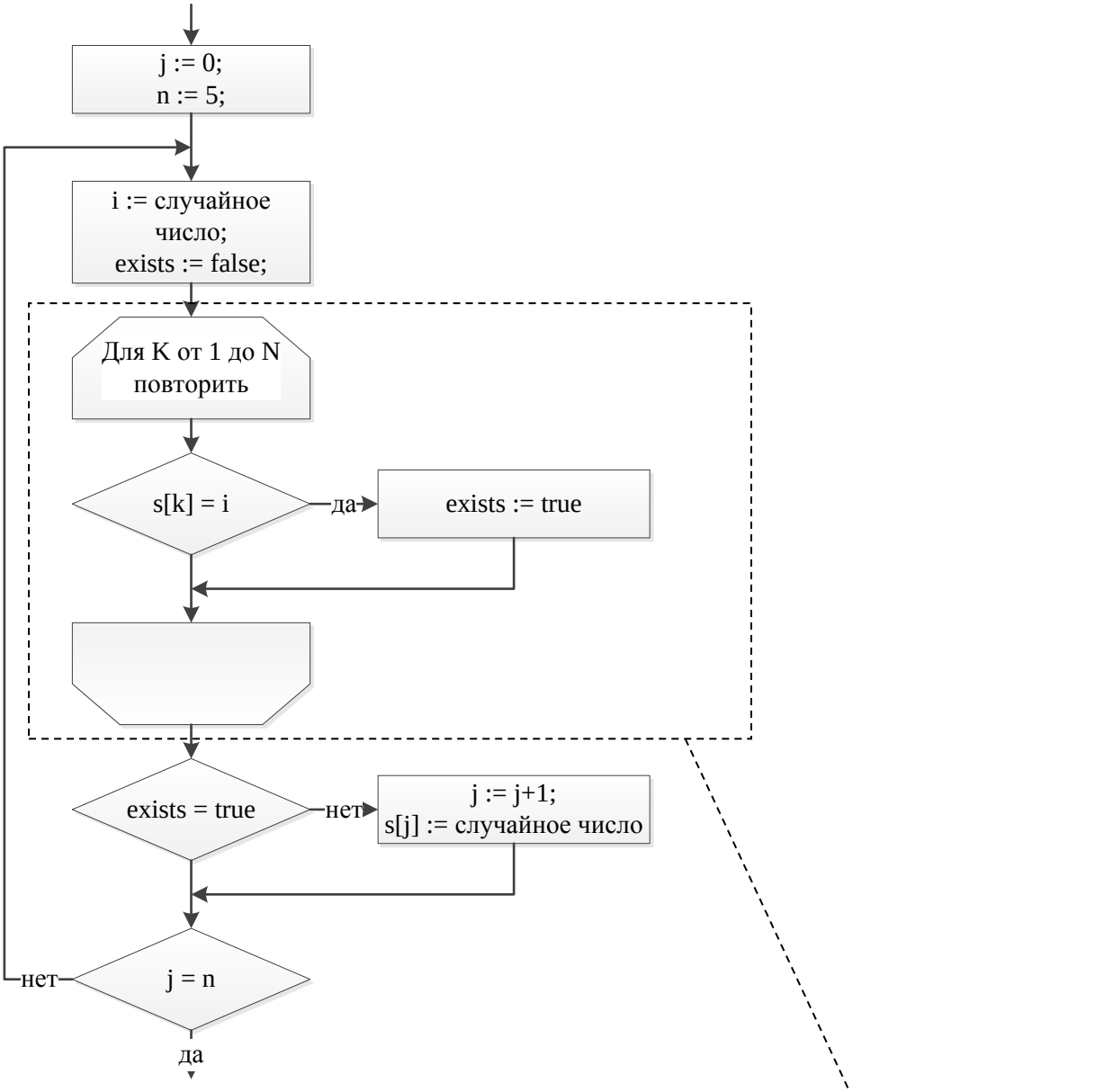
2-й случай: Множества используются для *формирования неповторяющегося набора элементов*.

ПРИМЕР. Заполнить массив из *5* элементов пятью случайными неповторяющимися числами из некоторого диапазона. Без использования множеств это можно сделать следующим образом:

```
Var
 s: array [1..5] of byte;  {массив для случайных чисел}
 i: byte;                  {случайное число}
 j: byte;                  {количество сформированных случайных чисел}
 k: byte;                  {параметр цикла}
 exist: boolean;           {случайное число уже есть в массиве}
begin
 {Начальное заполнение}
 fillchar (s, 5, 0);  j := 0;
 {инициализация датчика случайных чисел}
 randomize;
 {цикл, пока не заполним}
repeat
 i :=  random (?);
 exist := false;
 {проверка вхождения в набор}
 for   k := 1  to  5  do
  begin
   if (s [k] = i) and (i <> 0)
    then exist := true;
  end;
 {смотрим результаты анализа}
 if (exist = false) and (i <> 0)
  then begin
        inc (j); s[j] := i;
       end;
until j = 5;
end.
```

На "?" нужно поставить число, которое должно быть верхней границей диапазона случайных чисел.

что будет, если эту проверку убрать?(см. ниже)



***Замечание**: Random(10) возвращает целое число в диапазоне 0 <= x < 10*

Если все оставить как есть, то ноль также будет включен в число элементов формируемого массива. Для его исключения необходимо добавить проверку: *(i <> 0)*.

С использованием множеств та же задача решается следующим образом (более кратко):

```
Var
 a: array[1..5] of integer; {массив - носитель формируемых чисел}
 s: set of byte;      {вспомогательное множество}
 i: byte;             {случайное число }
 j: byte;             {количество сформированных случайных чисел}
begin
 {Начальное заполнение}
 s := [ ];
 j := 0;
fillchar(a, 5*sizeof(integer), 0);
 {инициализация датчика случайных чисел}
 randomize;
 {цикл, пока не заполним}
 repeat
  i :=  random (?);
  if (not (i in s)) and (i <> 0)  {Проверка вхождения в набор}
   then
    begin
     s := s + [i];
     inc (j);
     a[j]:=i;
    end;
 until j = 5;
end.
```


Видно, что с использованием множеств та же самая задача решается без использования внутреннего цикла for и без сложных дополнительных проверок.

Примечание. Как и в предыдущем случае, используется процедура *inc*:

inc (j); → *j := j +1.*

Использование этой процедуры влияет на ситуацию выхода из цикла.

Если множество сформировано и необходимо, например, **вывести элементы множества** на экран (печать), это можно сделать следующим образом.

Идея этого способа состоит в следующем:

- вводится вспомогательная переменная , которая в цикле принимает все возможные значения базового типа множества;
- если текущее значение этой переменой входит в множество, то это значение вспомогательной переменной выводится на экран.

```
Var
 s : set of 1 .. 100;
 k : 1 .. 100;              {вспомогательная переменная}
begin
 {формирование множества}
  ...
 {цикл по вспомогательной переменной}
 for k :=  1 to 100 do
  if k in s then
   writeln (k,' входит в s');
end.
```

Элементы множества будут распечатаны в том **порядке, в котором "пробегает" свои значения переменная *K*.**

Далее - Вспомогательные алгоритмы.