

МНОЖЕСТВА

Диапазонный, или
интервальный тип

Объявление множеств

- В языке программирования Pascal существует понятие множества, имеющее смысл некоторого собрания элементов, одно и того же базового типа. Базовый тип определяет перечень всех элементов, которые вообще могут содержаться в данном множестве.

продолжение

- В качестве базового типа может выступать любой простой порядковый тип. Но вещественные числа (**real** не порядковый тип) и строки (не простой и не порядковый тип) не могут быть элементами множества.

продолжение

- Размер множества в Turbo Pascal всегда ограничен некоторым предельно допустимым количеством элементов. Во множествах допускаются только такие элементы, порядковые значения которых не выходят за границы 0..255. Для целочисленных множеств это означает, что в них могут присутствовать только числа от 0 до 255. Отрицательные элементы множеств в Turbo Pascal не допускаются. Поэтому базовыми типами не могут быть типы **shortint**, **integer**, **longint**.

продолжение

- Если же необходимо множество целочисленных объектов, то базовый тип должен объявлен как диапазон типа **byte**. Для множеств, содержащих символы, подобных затруднений нет, поскольку базовым типом для них является **char** (а в нем 256 значений с порядковыми номерами от 0 до 255).

продолжение

- В математике для обозначения множества используют фигурные скобки (например, $\{4, 7, 12\}$), в Паскаль — квадратные (например, $[1, 3, 5]$). Порядок элементов во множестве не имеет значения. Так, записав $[3, 6, 9]$ или $[9, 3, 6]$, мы будем иметь дело с одним и тем же множеством. Более того, многократное повторение одного и того же элемента не меняет множество. Например, $[4, 7, 3]$ и $[3, 7, 4, 4]$ — это одно и то же множество.

продолжение

- По форме записи объявление переменной типа множество сходно с объявлением одномерного массива:
- **var** имя: **set of** тип;

продолжение

- Например, объявление переменной `ch`, рассматриваемой как множество с базовым типом **char**, имеет вид:
- **var ch: set of char;**

продолжение

- В отличие от элементов массива, элементы множества не упорядочены и не имеют индексов.
- Можно сначала объявить тип множества, а потом использовать его для объявления переменных:
- **type** t_ch = **set of char**; **var** ch1, ch2:
t_ch;

продолжение

- Довольно часто в качестве базового типа множества используется тип перечисления или некоторый его диапазон:
- **type** week_days = (Mon, Tue, Wed, Thu, Fri);
var work_days: **set of** week_days; lett: **set of** 'A'..'Z';
- Объявление переменной-множества не дает ей определенного значения.

Построение множества

- Чтобы во множестве появились элементы, необходимо выполнить оператор присваивания, в левой части которого стоит имя переменной-множества, а в правой — конструктор множества или некоторое выражение над множествами.

Конструктор множества

- *Конструктор множества* — это заключенный в квадратные скобки перечень элементов, разделенных запятыми. В качестве элементов могут использоваться диапазоны значений:
- **type** week_days = (Mon, Tue, Wed, Thu, Fri);
var work_days: **set of** week_days; lett: **set of** 'A'..'Z'; **begin** work_days := [Mon, Wed, Thu];
lett := ['C', 'E'..'M', 'Z'] **end.**

продолжение

- Следует помнить, что при задании множества порядок его элементов безразличен, но при задании диапазона такой порядок важен.
- Множество, в котором нет элементов, называется пустым (или нуль-множеством). В языке программирования Паскаль обозначается квадратными скобками, между которыми нет элементов:

■ `work_days := [];`

продолжение

- Множество может быть объявлено типизированной константой, для чего в описании после знака равенства следует указать конструктор множества. Например:

const lett: **set of** ['a'..'я'] = ['a', 'e', 'и', 'o', 'y', 'ы', 'э', 'ю', 'я'];

продолжение

- Конструируя множества, можно использовать и переменные при условии, что их текущие значения попадают в диапазон базового типа множества. Так, если `ch1` и `ch2` имеют тип **char**, то допустима следующая последовательность операторов:
 - `ch1 := 'A'; ch2 := 'K'; chs := [ch1, ch2, 'M'];`
 - В результате получится множество ['A', 'K', 'M']

продолжение

- Элементы множества нельзя вводить и выводить. Для организации ввода-вывода элементов множества следует использовать вспомогательные переменные. В то же время можно использовать множества как элементы типизированных файлов.

Действия над множествами

- Объединение, пересечение и разность множеств
- Над множествами выполнимы объединение (+), пересечение (*) и разность (-).
- *Объединение* двух множеств A и B ($A + B$) – это новое множество, состоящее из элементов, принадлежащих множеству A или B , либо тому и другому одновременно.

пример

- **var** chs1, chs2, chs3: **set of char**;
- **begin**
- chs1 := ['a', 'b', 'd']; chs2 := ['m', 'd', 'e'];
chs3 := chs1 + chs2 + ['k', 'n'];
- **end.**
- Результат: chs3 = ['a', 'b', 'd', 'm', 'e', 'k', 'n'].

продолжение

- *Пересечение* двух множеств A и B ($A * B$) – это множество, состоящее из элементов, одновременно принадлежащих множествам A и B .
- $chs3 := chs1 * chs2;$
- Результат: $chs3 = ['d']$.

продолжение

- *Разность* двух множеств A и B ($A - B$) – это новое множество, состоящее из элементов множества A , не вошедших в множество B .
- $chs1 := ['a', 'e', 't']; chs2 := chs1 - ['e'] \{$
 $['a', 't'] \}$ $chs3 := ['m', 'n', 't'] - chs2 \{$
 $['m', 'n'] \}$

продолжение

- Манипулируя операциями над множествами, можно добавлять элементы к множествам или удалять их.
- Для вставки и удаления элементов при работе с множествами в Pascal введены две процедуры:
- `include (имя_множества, элемент)` `exclude (имя_множества, элемент)` Первая из них позволяет выполнить добавление одного элемента в указанное множество, а вторая удалить.

пример:

- `include (chs1, 'g');` { *аналогично chs1 + ['g']* }
- `exclude (chs2, 'a');` { *аналогично chs2 - ['a']* }

Другие операции над множествами

- Над множествами можно выполнять четыре операции сравнения: $=$, \neq , \supseteq , \subseteq .
- Два множества A и B равны ($A = B$), если каждый элемент множества A является элементом множества B и наоборот.
- Два множества A и B не равны ($A \neq B$), если они отличаются хотя бы одним элементом.
- Множество A является подмножеством множества B ($A \subseteq B$, или $B \supseteq A$), если каждый элемент из A присутствует в B .

продолжение

- Имеется также возможность выяснить, принадлежит ли данный элемент некоторому множеству. Для этого служит операция **in**. Пусть A – множество элементов некоторого базового типа, а x – переменная (константа, выражение) этого типа. Тогда выражение $x \text{ in } A$ истинно, если значение x является элементом множества A .
- Все операции сравнения множеств, а также операция **in** возвращают логическое значение **true** или **false**.

продолжение

- В сложных выражениях над множествами операции имеют следующие приоритеты:
 - *
 - +, -
 - =, <>, <=, >=, in

Общий вид:

- Для переменной скалярного (перечисляемого) типа можно указать некоторое подмножество значений, которые может принимать данная переменная.

СИНТАКСИС

- $a: \min..max$;
- здесь a – интервальная переменная, \min – левая граница, max – правая граница подмножества (диапазона). Границы диапазона разделяются двумя точками; граница \min всегда должна быть меньше max .

продолжение

- Константы `min` и `max` должны принадлежать одному и тому же типу. Они определяют базовый тип переменной `a`. Так, если границы являются целыми числами типа **`integer`**, то под переменную `a` будет выделен такой же объем памяти, что и под тип **`integer`**. Однако переменная `a` сможет принимать только те значения, которые определены границами ее диапазона.

Примеры

- Пусть переменная *k* должна принимать значения из множества `-1000..1000`. Тогда ее следует объявить как `k: -1000..1000`. При этом базовым типом переменной *k* является тип **`integer`**, т.к. границами диапазона являются целые константы `-1000` и `1000`.
- Если переменная *b* может принимать одно из значений `red`, `yellow`, `green`, то эту переменную можно описать так: `b: red..green`; базовым типом для *b* является тип `color`:

Пример 1

- **type** color=(red,yellow,green,blue); **var** b:red..green; **begin** b:=red; writeln(b); b:=yellow; writeln(b); b:=green; writeln(b); readln **end**.

продолжение

- Пусть i – переменная, принимающая значения года рождения сотрудника какого-либо учреждения. Имеет смысл ограничить диапазон значений i подмножеством, т.е. описать примерно так: $i: 1930\dots2000$

Множества в примерах

- Пример. Пусть в вашем распоряжении имеется множество из трех монет разного достоинства: 1 р, 5 р, 10 р. Из этих монет можно составить следующие подмножества (их число равно $2^3 = 8$):
 - {1};
 - {5};
 - {10};
 - {1, 5};
 - {1, 10};
 - {5, 10};
 - {1, 5, 10};
 - {}

продолжение

- Эти подмножества и будут принадлежать некоторому множеству, тип которого назовем `sum`. Сами элементы (монеты), из которых составляется подмножество, пусть принадлежат некоторому базовому типу, который назовем `monet`.
Опишем типы данных этого примера:

продолжение

- **type** monet = (m1, m5, m10);
- sum = set of monet;

Пример 2

- Рассмотрим в качестве элементов базового типа сигналы от 4-х абонентов (ab1, ab2, ab3, ab4), поступающие на телефонную станцию. Обозначим базовый тип через abonent:
- `type abonent = (ab1, ab2, ab3, ab4),`

продолжение

- тогда комбинации сигналов можно описать переменной типа **множество**. Назовем этот тип `sing`:
- `sing = set of abonent;`

продолжение

- Тип `sing` описывает 16 комбинаций.
- В общем виде тип множество описывается так:
- `type a = set of tc;` здесь `a` – идентификатор типа (произвольный); `tc` – тип компонент множества называемый базовым типом.

продолжение

- Значение переменной типа множество изображается путем перечисления конкретных компонентов, разделенных запятыми и заключенных в квадратные скобки.
- Пример. Пусть базовый тип `int` и тип `a` заданы так:
- `type int = 1..3; a = set of int;`

- Переменная a в этом случае может принимать восемь значений: $[1]$, $[2]$, $[3]$, $[1, 2]$, $[1, 3]$, $[2, 3]$, $[1, 2, 3]$, $[\]$. Например, если переменная b имеет тип a , то можно присвоить ей одно из перечисленных выше значений: $b := [1, 3]$; $b := [1, 3, 2]$; и т.д.
-

пример

- Если базовый тип описывает набор двоичных бит, то можно получить их комбинацию. Пусть
- `type bin = (bit1, bit2, bit3);` `bts = set of bin;`

продолжение

- Переменная типа `bts` может принимать восемь значений.
Таким образом, используя переменные типа **`set`**, можно работать с битовой информацией.

продолжение

- В качестве базового типа может использоваться любой простой тип, кроме **real**. Если задача требует использования множества, состоящего из большого числа элементов, то его можно представить как массив множеств, состоящих из допустимого числа элементов.



Данные типа `set`

- Данные типа **`set`** задаются путем перечисления значений, разделенных запятыми и заключенных в квадратные скобки.
- Общий вид:
- `[expr1, expr2, ...exprn]`; Здесь `expri` – выражение базового типа.

продолжение

- Порядок следования выражений несущественен. Непустой набор может быть также выражением вида:
- $[expr1]; [expr1..exprk]; [expr1, exprk..exprn];$

продолжение

- Данные вида $[expr1..exprk]$ соответствуют набору всех элементов базового типа от значения $expr1$ до $exprk$.
- Пример. $[3 * 6 - 7..15 + 4]$ соответствует набору $[11..19]$, т.е. $[11, 12, 13, 14, 15, 16, 17, 18, 19]$.

продолжение

- Если окажется, что для $[i..j]$, $i > j$, то такое множество интерпретируется как пустое, а в случае $i = j$ – как множество, содержащее один элемент – i .
- Пример. $[3 * 6 - 7..5 + 6]$ эквивалентно $[11]$.

Пример 3

- `type color = (red, yellow, green, blue);`
- `var mix: set of color;`
-
- `mix := [red, blue];`

Пример 4

- type n = (1, 3, 5, 7, 9);
- var k: set of n;
-
- k := [3..9];
- В этом случае в k запишется комбинация [3, 5, 7, 9];

Операции над множествами

- К переменным типа **set** применимы следующие операции:
- $=$,
- $\langle \rangle$,
- $>=$,
- \leq ,
- in ,
- $+$, $-$, $*$.

продолжение

- Операции = и <> используются для проверки эквивалентности: два значения переменной типа **set** считаются равными, если они состоят из одних и тех же элементов.

Пример 5

- $[1, 3] = [3, 1]$ возвращает **true**,
- $[1..3] = [1, 2, 3]$ возвращает true,
- $[1] <> [2]$ возвращает true,
- $[1, 2, 3] = [1, 4, 3]$ возвращает false,
- $[\text{red}, \text{blue}] = [\text{red}, \text{yellow}]$ возвращает false.

продолжение

- Операции \geq и \leq используются для проверки принадлежности одного множества другому: так, если множество a содержится во множестве b , то $a \leq b$ дает true.

Пример 6

- $[1, 2] \leq [1, 2, 3]$ дает true
- пустое множество $[\]$ содержится во всех множествах, т.е. всегда $[\] \leq [b]$ дает true.

продолжение

- Операция **in** используется для установления наличия определенного элемента в величине типа **set**. Так, если x есть элемент множества b , то $(x \text{ in } b)$ дает `true`. Общий вид:
- $x \text{ in } a$; здесь x – величина базового типа, a – величина типа `set`.

Пример 7

- `red in [red, yellow]` возвращает `true`; `red in [blue, green]` возвращает `false`.
Замечание 1. Чтобы проверить, является ли значение `n` цифрой, удобно использовать операцию `in` следующим образом:
 - `if n in [0..9] then ...`
- Замечание 2. Результат операции `in` может быть неопределенным в некоторых случаях.

Пример 8

- $[1, 3] + [1, 4] = [1, 3, 4]$; $[1, 3] * [1, 4] = [1]$; $[1, 3] - [1, 4] = [3]$. Операция $a := a + x$ добавляет элемент x к множеству a . Если x уже имелся в a , то множество a не меняется. $a := a - x$ исключает x из a . Если x отсутствовал в a , то множество a не меняется.