Перечисляемые типы данных

**Перечисляемый тип** задается непосредственно перечислением всех значений, которые может принимать переменная данного типа. Отдельные значения указываются через запятую, а весь список зак­лючается в круглые скобки.

type

TDirection = (North, South, East, West);

На базе типа TDirection можно объявить переменную Direction и присвоить ей значение:

var

Direction: TDirection;

begin

Direction := North;

end.

На самом деле за идентификаторами значений перечисляемого типа стоят целочисленные константы. По умолчанию, первая константа равна 0, вторая — 1 и т.д. Существует возможность явно назначить значения идентификаторам:

type

TSizeUnit = (Byte = 1, Kilobyte = 1024 \* Byte, Megabyte = Kilobyte \* 1024,

Gigabyte = Megabyte \* 1024);

Выражения и константы перечисляемого типа допустимы для использования в операторе Case. Операции отношения и логи­ческие операции допустимы для значений перечисления одного и то­го же типа. Упорядочение осуществляется по номеру элемента в описании типа.

Для работы с данными перечисляемого типа предназначены стандартные подпрограммы Succ, Pred, Ord.

Pred(S) - возвращает элемент, предшествующий S в списке зна­чений типа. Тип результата совпадает с типом параметра. Если предшествующего S элемента не существует, возникает программное прерывание.

***Пример.*** Write (Pred(90)); Результат =89.

Succ(S) - возвращает значение, следующее за S в списке значе­ний типа. Тип результата совпадает с типом параметра. Если сле­дующее за S значение отсутствует, возникает программное прерыва­ние.

***Пример.*** Write (Succ(90)); Результат *--* 91.

Ord(S) - возвращает порядковый номер значения S в множестве, определенном типом S. Результат целочисленного типа. Если параметр имеет тип integer, то это значение функции выдается как результат. Если параметр любого другого перечисляемого типа, то результат – порядковый номер, определяемый отображением значений данного типа на последовательность неотрицательных чисел, начиная с нуля.

Множества

Объявление множества

*Множество* — это составной тип данных для представления набора некоторых элементов как единого целого. Область значений множества — набор всевозможных подмножеств, составленных из его элементов. Все элементы множества должны принадлежать однобайтовому порядковому типу. Этот тип называется *базовым типом множества*.

Для описания множественного типа используется словосочетание **set of**, после которого записывается базовый тип множества:

type

TLetters = set of 'A'..'Z';

Теперь можно объявить переменную множественного типа:

var

Letters: TLetters;

Можно объявить множество и без предварительного описания типа:

var

Symbols: set of Char;

В выражениях значения элементов множества указываются в квадратных скобках: [2, 3, 5, 7], [1..9], ['A', 'B', 'C']. Если множество не имеет элементов, оно называется пустым и обозначается как [ ]. Пример инициализации множеств:

const

Vowels: TLetters = ['A', 'E', 'I', 'O', 'U'];

begin

Letters := ['A', 'B', 'C'];

Symbols := [ ]; { пустое множество }

end;

Количество элементов множества называется *мощностью*. Мощность множества в языке Delphi не может превышать 256.

Операции над множествами

При работе с множествами допускается использование операций отношения (=, <>, >=, <=), объединения, пересечения, разности множеств и операции **in**.

*Операции сравнения* (=, <>). Два множества считаются равными, если они состоят из одних и тех же элементов. Порядок следования элементов в сравниваемых множествах значения не имеет. Два множества A и B считаются не равными, если они отличаются по мощности или по значению хотя бы одного элемента.

|  |  |
| --- | --- |
| *Выражение* | *Результат* |
| [1, 2] <> [1, 2, 3] | True |
| [1, 2] = [1, 2, 2] | True |
| [1, 2, 3] = [3, 2, 1] | True |
| [1, 2, 3] = [1..3] | True |

*Операции принадлежности* (>=, <=). Выражение A >= B равно True, если все элементы множества B содержатся в множестве A. Выражение A <= B равно True, если выполняется обратное условие, т.е. все элементы множества A содержатся в множестве B.

|  |  |
| --- | --- |
| *Выражение* | *Результат* |
| [1, 2] <= [1, 2, 3] | True |
| [1, 2, 3] >= [1, 2] | True |
| [1, 2] <= [1, 3] | False |

*Операция* **in**. Используется для проверки принадлежности элемента указанному множеству. Обычно применяется в условных операторах.

|  |  |
| --- | --- |
| *Выражение* | *Результат* |
| 5 in [1..9] | True |
| 5 in [1..4, 6..9] | False |

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

if (X = 1) or (X = 2) or (X = 3) or (X = 5) or (X = 7) then

можно заменить более коротким:

if X in [1..3, 5, 7] then

Операцию **in** иногда пытаются записать с отрицанием: X **not** **in** S. Такая запись является ошибочной, так как две операции следуют подряд. Правильная запись имеет вид: **not** (X **in** S).

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

|  |  |
| --- | --- |
| *Выражение* | *Результат* |
| [ ] + [1, 2] | [1, 2] |
| [1, 2] + [2, 3, 4] | [1, 2, 3, 4] |

*Пересечение множеств* (\*). Пересечение двух множеств — это третье множество, которое содержит элементы, входящие одновременно в оба множества.

|  |  |
| --- | --- |
| *Выражение* | *Результат* |
| [ ] \* [1, 2] | [ ] |
| [1, 2] \* [2, 3, 4] | [2] |

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

|  |  |
| --- | --- |
| *Выражение* | *Результат* |
| [1, 2, 3] – [2, 3] | [1] |
| [1, 2, 3] – [ ] | [1, 2, 3] |

В язык Delphi введены две стандартные процедуры Include и Exclude, которые предназначены для работы с множествами.

Процедура **Include**(S, I) включает в множество S элемент I. Она дублирует операцию + (плюс) с той лишь разницей, что при каждом обращении включает только один элемент и делает это более эффективно.

Процедура **Exclude**(S, I) исключает из множества S элемент I. Она дублирует операцию – (минус) с той лишь разницей, что при каждом обращении исключает только один элемент и делает это более эффективно.

|  |  |
| --- | --- |
| *Выражение* | *Результат* |
| S := [1, 3]; | [1, 3] |
| Include(S, 2); | [1, 2, 3] |
| Exclude(S, 3) | [1, 2] |

Использование в программе множеств дает ряд преимуществ: значительно упрощаются сложные операторы **if**, улучшается наглядность программы и понимание алгоритма решения задачи, экономится время разработки программы. Поэтому множества широко используются в библиотеке компонентов среды Delphi.