**Множества. Процедурыне типы.**

Понятие ***множества*** в математике является одним из основных и обозначает некоторую неупорядоченную совокупность объектов, которые называются *элементами* множества. Например, множество простых чисел, множество жильцов одной улицы, множество букв алфавита и т.д. Из приведенных примеров видно, что множества могут содержать *подмножества*(множество жильцов одной улицы состоит из подмножеств людей, живущих в разных домах на этой улице).

Следует обратить внимание на **две основных особенности множеств**:

* во множестве могут содержаться элементы только одного, *базового* типа (например, множество простых чисел не может содержать еще и буквы);
* порядок элементов множества не фиксируется.

Например, такая совокупность элементов {1, 2, abc, ?!?} вообще не считается множеством, а совокупности {1, 2, 5, 8} и {8, 1, 5, 2} – эквивалентные множества.

Для представления такого типа данных и для реализации некоторых моментов математического аппарата теории множеств в Паскале существует ***множественный тип данных*** (множества).

***Множеством*** *называется совокупность однотипных элементов, рассматриваемых как единое целое*. В Паскале могут быть только конечные множества. Число элементов исходного множества в Турбо Паскале не может быть больше **256**, а порядковые номера элементов (т.е. значения функции Ord) должны находиться в пределах от **0**до **255**.

В отличие от элементов массива элементы множества не пронумерованы, не упорядочены. Каждый отдельный элемент множества не идентифицируется, и с ним нельзя выполнить какие-либо действия. Действия могут выполняться только над множеством в целом.

Тип элементов множества называется *базовым* типом. Базовый тип может быть любой простой тип (стандартный, перечисляемый или ограниченный), за исключением типов *Integer* и *Real,* т.к. в этом случае мощность множества будет превышать 256.

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

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

**Конструктор множества**

Конкретные значения множества задаются с помощью ***конструктора множества***, представляющего собой список элементов, заключенный в квадратные скобки. Сами элементы могут быть либо константами, либо выражениями базового типа.

Примеры задания множеств с помощью конструктора:

[3, 4, 7, 9, 12] – множество из пяти целых чисел;

[1..100] – множество целых чисел от 1 до 100;

[‘a’, ‘b’, ‘c’] – множество, содержащее три литеры *a, b, c*;

[‘A’..’Z’, ‘?’, ‘!’] – множество, содержащее все прописные латинские буквы, а также знаки ? и !.

Символы **[]** обозначают пустое множество, т.е. множество, не содержащее никаких элементов.

Не имеет значения порядок записи элементов множества внутри конструктора. Например, [1, 2, 3] и [3, 2, 1] эквивалентные множества.

Каждый   элемент   во   множестве  учитывается  только  один  раз.   Пример:   множество  [1, 2, 3, 4, 5]  эквивалентно [1..5].

**Описание множеств**

Для задания типа множества используются зарезервированные слова ***Set*** и ***Of***, а затем указываются элементы этого множества, как правило, в виде перечисления или диапазона.

 Множества могут быть описаны двумя способами:

* **Type** ***имя\_типа* =** **Set Of** ***базовый тип*;**

**Var** ***имя\_множества*: *имя\_типа*;**

* **Var** ***имя\_множества*:** **Set Of** ***базовый тип*;**

Здесь ***базовый тип*** — тип элементов, входящих во множество.

Пример 1:

* Type
* digits = Set Of 1..5;
* Var
* s: digits;

Пример 2:

1. Type
2. elemcolor = (red, yellow, blue);
3. color = Set Of elemcolor;

Пример 3:

* Var A, D: Set Of Byte;
* B: Set Of ?a? .. ?z?;
* C: Set Of Boolean;

Нельзя вводить значения во множественную переменную оператором ввода и выводить оператором вывода. Множественная переменная может получить конкретное значение только в результате выполнения оператора присваивания следующего формата:

**< множественная переменная > := < множественное выражение >**

Пример:

* A := [50, 100, 150, 200];
* B := [?m?, ?n?, ? k?];
* C := [True, False];
* D := A;

Кроме того, выражения могут включать в себя операции над множествами.

**Операции над множествами**

В Паскале реализованы основные операции теории множеств. Это ***объединение, пересечение, разность*** множеств. Во всех таких операциях операнды и результаты есть множественные величины одинакового базового типа.

* ***Объединение множеств***. Объединением двух множеств ***A*** и ***B*** называется множество, состоящее из всех элементов, принадлежащих хотя бы одному из множеств ***A*** и ***B***. Знак операции объединения в Паскале **+**.

Пример:          [1, 2, 3, 4] + [3, 4, 5, 6] ? [1, 2, 3, 4, 5, 6]

* ***Пересечение множеств***. Пересечением двух множеств ***A****и****B*** называется множество, состоящее из всех элементов принадлежащих одновременно множеству ***A*** и множеству ***B***. Знак операции пересечения в Паскале \*.

Пример:          [1, 2, 3, 4] ***\**** [3, 4, 5, 6] ? [3, 4]

* ***Разность множеств***. Разностью двух множеств ***A*** и ***B*** называется множество, состоящее из элементов множества ***A***, не принадлежащих множеству ***B***. Знак операции пересечения в Паскале ***—***.

Пример:          [1, 2, 3, 4] ***—*** [3, 4, 5, 6] ? [1, 2]

                        [3, 4, 5, 6] ***—*** [1, 2, 3, 4] ? [5, 6]

Очевидно, что операции объединения и пересечения – перестановочны, а разность множеств – не перестановочная операция.

* ***Операции отношения***. Множества можно сравнивать между собой, т.е. для них определены операции отношения. Результатом отношения является логическая величина *true* или *false*. Для множеств применимы все операции отношения, за исключением > и <. Ниже в таблице описаны операции отношения над множествами (множества ***A*** и ***B*** содержат элементы одного типа).

|  |  |
| --- | --- |
| **Результат** | |
| ***True*** | ***False*** |
| **A = B** | Множества ***A*** и ***B*** совпадают | В противном случае |
| ***A*** **<>** ***B*** | Множества ***A*** и ***B*** не совпадают | В противном случае |
| ***A*** **<=** ***B*** | Все элементы ***A*** принадлежат ***B*** | В противном случае |
| ***A*** **>=** ***B*** | Все элементы ***B*** принадлежат ***A*** | В противном случае |

Примеры:

* Var M: Set Of Byte;
* M:=[3, 4, 7, 9];

Тогда операции отношения дадут следующие результаты:

|  |  |
| --- | --- |
| **Операция** | **Результат** |
| M=[4, 7, 3, 3, 9] | *true* |
| M<>[7, 4, 3, 9] | *false* |
| [3, 4]<=M | *true* |
| []<=M | *true* |
| M>=[1..10] | *false* |
| M<=[3..9] | *true* |

* ***Операция вхождения***. Эта операция устанавливает связь между множеством и скалярной величиной, тип которой совпадает с базовым типом множества. Если ***x*** – такая скалярная величина, а ***M*** – множество, то операция вхождения записывается так: ***x*** **In** ***M***. Результат – логическая величина *true*, если значение ***x*** входит в множество  ***M***, и *false* — в противном случае.

Пример для описанного выше множества:  **4** **In** ***M***— *true*, **5** **In** ***M***– *false*.

**Процедурные типы**

Перевод из справочной системы Delphi

Процедурные типы позволяют вам работать с процедурами и функциями как со значениями, которые могут быть присвоены переменным или переданы другим процедурам и функциям в качестве параметров.

О процедурных типах

Следующий пример демонстрирует использование процедурного типа. Предположим, вы объявили функцию с именем Calc, которая принимает в качестве параметров два целочисленных значения и возвращает одно:

function Calc(X,Y: Integer): Integer;

Вы можете связать функцию Calc с переменной F:

var F: function(X,Y: Integer): Integer;

F := Calc;

Если вы при объявлении процедуры или функции удалите идентификатор, идущий после слов procedure или function, то, что остается слева может быть правой частью объявления процедурного типа. Вы можете использовать такие имена типов непосредственно при объявлении переменных (как в предыдущем примере) или при объявлении новых типов:

type

TIntegerFunction = function: Integer;

TProcedure = procedure;

TStrProc = procedure(const S: string);

TMathFunc = function(X: Double): Double;

var

F: TIntegerFunction; // F – это функция без параметров,

//которая возвращает integer

Proc: TProcedure;

// Proc – процедура без параметров

SP: TStrProc; // SP – это процедура,

//которая принимает один строковый параметр

M: TMathFunc; // M – это функция,

//которая принимает параметр типа Double (real)

// и возвращает Double

procedure FuncProc(P: TIntegerFunction);

// FuncProc – процедура, которая принимает

// один параметр – функцию, которая

// возвращает значение integer

Для платформы Win32 переменные, указанные в предыдущем примере, являются процедурными указателями (указателями, хранящими адрес процедур или функций). Если вы хотите обратиться к методам объекта, вам необходимо добавить в объявление процедуры или функции слова of object. То есть:

type

TMethod = procedure of object;

TNotifyEvent = procedure(Sender: TObject) of object;

Эти типы представляют собой указатели на методы. Указатели на методы на самом деле являются парой указателей, первый из которых хранит адрес метода, а второй – ссылку на объект, которому принадлежит метод. Давая объявление вида:

type

TNotifyEvent = procedure(Sender: TObject) of object;

TMainForm = class(TForm)

procedure ButtonClick(Sender: TObject);

...

end;

var

MainForm: TMainForm;

OnClick: TNotifyEvent;

Мы можем выполнить следующую инструкцию:

OnClick := MainForm.ButtonClick;

Два процедурных типа совместимы, если они имеют:

Одинаковую конвенцию вызова

Одинаковое возвращаемое значение (или не имеют такового)

Одинаковое количество параметров

Параметры на одних и тех же позициях имеют одинаковый тип (имена параметров не имеют значения).

Процедурные указатели всегда несовместимы с указателями на методы объектов. Значение nil может быть присвоено переменной любого процедурного типа.

Вложенные процедуры и функции (подпрограммы, объявленные внутри других подпрограмм) не могут быть использованы в качестве процедурных значений или в качестве предопределенных процедур и функций. Если вы хотите использовать предопределенные подпрограммы (такие как Length) в качестве процедурных значений – используйте "обертку" вида:

function FLength(S: string): Integer;

begin

Result := Length(S);

end;

Процедурные типы в инструкциях и выражениях

Когда процедурная переменная находится в левой части инструкции присваивания, - компилятор ожидает обнаружить процедурное значение в правой части инструкции. Присваивание связывает переменную в левой части инструкции с функцией или процедурой, находящейся в правой части. В прочих контекстах, обращение к процедурной переменной выполняет вызов процедуры или функции, на которую она ссылается. Так же вы можете использовать процедурную переменную для передачи параметров:

var

F: function(X: Integer): Integer;

I: Integer;

function SomeFunction(X: Integer): Integer;

...

F := SomeFunction; // связывает SomeFunction с F

I := F(4); // вызывает функцию;

//и присваивает результат переменной I

В инструкциях присваивания тип переменной в левой части определяет интерпретацию процедурных указателей или указателей на методы, которые указаны в правой части. Например:

var

F, G: function: Integer;

I: Integer;

function SomeFunction: Integer;

...

F := SomeFunction; // связывает SomeFunction с F

G := F; // копирует F в G

I := G; // вызывает функцию;

//присваивает результат переменной I

Первая инструкция присваивает процедурное значение переменной F. Вторая инструкция копирует это значение в другую переменную. Третья инструкция выполняет вызов функции, на которую ссылается G и присваивает результат переменной I. Поскольку I – это целочисленная, а не процедурная переменная, последняя инструкция присваивания фактически выполняет вызов функции (которая возвращает целое число).

В некоторых ситуациях не совсем понятно, как должна интерпретироваться процедурная переменная. Рассмотрим инструкцию:

if F = MyFunction then ...;

В этом случае упоминание F приводит к вызову функции. Компилятор вызывает функцию, на которую ссылается F, затем вызывает MyFunction, затем сравнивает результат. Правило следующее: каждый раз, когда процедурная переменная встречается внутри выражения, - она интерпретируется как вызов процедуры или функции. В случае, когда F ссылается на процедуру (которая не может иметь возвращаемого значения), или когда F ссылается на функцию, которая требует параметров, указанная инструкция вызовет ошибку компиляции. Для сравнения процедурных значений, хранящихся в F и MyFunction, следует сравнивать:

if @F = @MyFunction then ...;

@F преобразует F в нетипизированный указатель, содержащий адрес, а @MyFunction возвращает адрес MyFunction.

Чтобы получить адрес процедурной переменной, а не адрес который хранится в ней, необходимо использовать @@. Например, @@F возвращает адрес F.

Оператор @ может так же быть использован для связывания нетипизированного указателя с процедурной переменной. То есть:

var StrComp: function(Str1, Str2: PChar): Integer;

...

@StrComp := GetProcAddress(KernelHandle, 'lstrcmpi');

вызывает функцию GetProcAddress и указывает StrComp на результат.  
Любая процедурная переменная может принимать значение nil, что означает, что она не указывает никуда. При этом попытка вызова подпрограммы по указанию имени этой переменной вызовет ошибку. Для проверки значения процедурной переменной, следует пользоваться стандартной функцией Assigned:

if Assigned(OnClick) then OnClick(X);

Литература

https://coderbook.ru/pascal-лекция-№8/

<http://pascal-study.blogspot.com/2011/05/blog-post_2361.html>