**abstract unit** Day

**end**

**unit** WorkDay **extend** Day

**const is**

Mon, Tue, Wed, Thur, Fri

**end**

**end**

**unit** DayOff **extend** Day

**const is**

Sat, Sun

**end**

**end**

**unit** WeekDay **extend** WorkDay, DayOff

**end**

**use const** WorkDay, DayOff

d0: Day **is** Mon

d0 := Sat

d1: WorkDay | DayOff is Mon

d1 := Sat

N – число битов, выделенных для хранения Real.

m – число битов на мантиссу

o – число битов на порядок

m + n + 1 = N 1 бит на знак. 0 => +, 1 => -

maxReal = (2 \*\*m \* 10) \*\* (2\*\*o)

minReal =

Вот смотри, что ты предлагаешь

**val unti** Real

**init** (even, fractional: Integer)

**init** (mantissa, order: Integer; fuckTheSameInitNameDumymParameter: Bit)

**end**

a **is** Real.init (5, 5) // instead of a is 5.5

b **is** Real.init (4, 6, Bit.0b0) // instead of b is 4E6

**Регулярное выражение имеет вид:**

{ шаблон повторения } условие повторения

Любое регулярное выражение при обработке выдает последовательность сущностей через запятую – т.е. некоторый список. Т.е. это по сути внутренность кортежа.

**Виды “шаблон повторения”:**

{ a } - просто переменная

{123} – просто константа

{‘a’} {“string”}

{a | b | c} – т.е. повторятся могут разные сущности

Сущность, стоящая внутри фигурных скобочек будет повторятся.

Как она повторяется задается условием повторения или шагом внутри шаблона повторения

**Виды “условие повторения”:**

Целая константа – сколько раз повторяется {6} 3 – повторить 6 три раза дает список 6, 6, 6 – по сути это внутренность кортежа.

{a + b} <= x дает последовательность a, a+ b, a+2\*b, …. a+ i\*b <= x т.е. последовательность с шагом ограниченная значением х

{a} 0+ - неограниченная последовательность. Пустая или произвольной длины a или a, a или a, a, a и так далее

Строка это

{Character.minCharacter |.. Character.maxCharacter} 0+

такое регулярное выражение задет любую строку.

**val unit** Bit

**const is**

0b0, 0b1

**end**

**end**

**val unit** Bit [N: Integer] **is**

**const is**

{Bit.0b0 | Bit.0b1}N

**end**

**end**

**unit** String [N:Integer] **extend** AString, Array [Character, N] **is**

/// String with fixed length

**const is**

"{Character.minCharacter |.. Character.maxCharacter}N"

**end**

**end**

**unit String extend AString is**

/// Variable length String

**const is**

"{Character.minCharacter |.. Character.maxCharacter}0+"

**end**

**end**

Это полный вынос мозга? :-)

1. Введение
2. Две основные константы – основа мироздания.
3. Целые, логические и символьные константы.
4. Вещественные, строковые и битовые константы.
5. Общее понятие константных объектов.
6. Как константные объекты заменяют перечислимые типы.
7. Заключение.

Перечислимые типы или константные объекты.

Понятие перечислимых типов позволяет поднять уровень абстракции на чуть более высокий уровень, чем просто пользование константами и следующий шаг – это понимание того что надо констант – это набор константных объектов некоторого типа. И в общем случае тип может быть любой. Таким образом, любой юнит на базе, которого порождается тип может задать все известные на момент описания юнита константные объекты данного юнит типа. Для целого типа это задается следующим примером

**val unit** Integer

**const**

Platform.minInteger **..** Platform.maxInteger

**end**

**end**

Таким образом, запись Integer.1 есть полное наименование константного объекта 1, для того чтобы не писать префикс имени юнита необходимо включить константы в тот юнит или фрагмент кода, где это необходимо

**abstract unit** Any **use const** Integer, Real, Boolean, Character, Bit, String

**end**

Вот такая конструкция use const позволяет использовать константные объекты из указанных юнитов без префиксов имен юнитов в данном юните и во всех его потомках. Таким образом, константы, к которым мы так привыкли можно использовать без префиксов в любом юните.

**unit** WeekDay

**const**

Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday

**end**

**end**

Теперь можно работать с константами типа WeekDay

day: WeekDay **is** WeekDay.Monday

**if** day **is**

WeekDay.Monday **..** WeekDay.Friday: StandardIO.putString (“Work day – go to the office!\n”)

WeekDay.Saturday, WeekDay.Sunday: StandardIO.putString (“WeekEnd – do what you like!\n”)

**end**

А для того, чтобы избежать обязательного префикса, достаточно написать **use const** WeekDay и после этого тот же самый пример станет короче, но не менее выразителен

**use const** WeekDay

day: WeekDay **is** Monday

**if** day **is**

Monday **..** Friday: StandardIO.putString (“Work day – go to the office!\n”)

Saturday, Sunday: StandardIO.putString (“WeekEnd – do what you like!\n”)

**end**

А так как юнит может иметь процедуру инициализации, то пример более общего случая будет выглядеть так

**unit** A

**const** A1.init, A2.init (T), A3.init (T1, T2) **end**

**init is end**

**init** (arg: T) **is end**

**init** (arg1: T1; arg2: T2) **is end**

**end**

И теперь уже можно использовать тип A и с константные объекты данного типа A1, A2 и A3.

Например, таким образом

**use const** A

foo (A1)

foo (argument: A) **is**

**if** argument

A1: // Некоторое действие(я)

A2 .. A3: // Другое действие(я)

**end**

**end**

Понятие константных объектов является более общим, чем перечислимые типы и позволяет поддержать привычный функционал перечислимых типов и распространить возможность сравнение по образцу (pattern matching) для любых типов.