Стандартные приемы Icon.

Начальные обозначения

- Обозначим некоторое выражение (вне зависимости от того, чем оно является) как expr.
- Не будем делать различия между элементарными выражениями и составными (если не оговорено иначе).
- Будем рассматривать типовые приемы работы с различными генерирующими выражениями.

Примитивы, работающие со структурами

- Предположим, что expr — это некоторая структура (число, строка, множество, список, файл и т.д)
- !expr — генерация элементов структуры. (В случае отсутствия контекста, сгенерирует самый первый элемент структуры)
- Существует несколько устойчивых выражений с подобным генератором.

Генерация элементов структуры

- Проход по всей структуре может быть осуществлен двумя способами.
- Способ 1. Итеративный без использования индексов:

```
every i := !l do {
    блок действий с переменной i
  }
```

Генерация элементов структуры

- Способ 2. Итеративный цикл без индексов:

```
while i := !l do {
блок действий с переменной i
}
```

- Иногда этот цикл требует every перед while. Поэтому следует очень внимательно следить за контекстом.

Генерация элементов с ограничениями

- Выражение every |expr генерирует все значения из expr, производит очистку и перезапуск выражения и процесс генерации возобновляется вновь.
- Дабы избежать бесконечной генерации используется ограничитель повторений:

|expr\n

где n — выражение задающее количество выдаваемых значений.

Бесконечные ряды значений

- Функция seq(i,j) предназначена для генерации бесконечных числовых рядов вида: i,i+j,i+2j,i+3j и т.д
- Может быть использована вместо проходов цикла по последовательностям, и в математических приложениях.

Случайные элементы структур

- Выражение ?expr генерирует случайный элемент из структуры и может быть использовано неограниченное число раз.
- Стоит помнить, что если expr дает числовое значение, то выражение ?expr дает случайное целое число от 1 до expr
- Данное выражение не работает для отрицательных чисел.

Другие манипуляции со структурами

- *expr — Размер (длина структуры).
- Используя это выражение можно реализовать следующий типовой проход по структуре используя индексы:
  every i := 1 to *expr+1 do {

  действия с expr[i]

}

Срезы структур

- Срезы или «слайсы» (slices) — выборочные выделения элементов из структур при помощи указания индексов.
- Базовый синтаксис срезов : expr[i:j]
  Это обозначает то, что из структуры берутся элементы с i-ого по j-ый.
- Допустимо использовать целые отрицательные значения. Положительные значения границ срезов обозначают что отсчет идет с левого элемента структуры, отрицательные (и ноль включительно) обозначают то, что отсчет идет с правого элемента структуры.

Часто встречающиеся варианты срезов

- Выражение expr[i:0] обозначает выборку всех элементов начиная с i-ого и заканчивая последним
- Выражение expr[i] (или избыточное expr[i:i]) обозначает захват i-ого элемента структуры
- Выражение expr[1:(*expr)] обозначает взятие всех элементов начиная с 1-ого и кончая предпоследним

Сокращенные варианты срезов

- Иногда для выборки определенного количества элементов из структуры очень удобно использовать сокращенные варианты срезов.
- Выражение expr[i+:j] обозначает выбор j элементов, начиная с i-ого и включая его. Фактически, эквивалентно конструкции expr[i:(i+j)]
- Выражение expr[i-:j] подобно упомянутому выше и эквивалентно конструкции expr[i-j:i] и также включает в себя i-ый элемент.

Формирование списков

- Список можно формировать из пустого списка по необходимости добавляя элементы в него:
  l := [] # формируем пустой список

put(l,expr) # затолкнет в конец списка структуру expr

- Также можно формировать списки просто, указывая элементы в них входящие:
  [expr1,expr2,...,exprn]
- Список с заданным количеством элементов можно сформировать так:
  list(n,expr) # сформирует список из n элементов

Типовые математические приемы

- Выражения типа expr := expr op x (где op — некий оператор, а x — свободная переменная) можно значительно сократить используя следующий синтаксис: expr op:= x
- В роли операторов могут выступать следующие действия: +,-,*,/,%,^,<,> и некоторые другие.
- Использование подобного синтаксиса позволяет сделать программы более краткими и понятными.

Отрицательные числа и степень.

- К сожалению, Icon не умеет возводить отрицательные числа в степень с помощью оператора ^.
- Для преодоления этой проблемы вставим в программу свою функцию возведения в степень:

```
 procedure pow(x,n)
if x>=0 then return x^n else {
if n%2=0 then return abs(x)^n else return
-(abs(x)^n)
```

Предустановленные константы

- В Icon имеется большой набор встроенных констант (которые называются почему-то keywords — ключевые слова)
- Математические константы:

&pi — число Пи

&e — Основание натуральных логарифмов

&phi — Золотое сечение

&digits — Все цифры

Принудительное преобразование типов.

- Иногда в программе требуется принудительное преобразование выражения к заданному типу. Для этого применяется следующий набор операторов: integer(expr), cset(expr), string(expr), numeric(expr), real(expr), set(expr).
- Кроме того, в разряд подобных функций можно внести еще две: char(o) — преобразует числовой код в символ и ord(c) — обратная операция к char().
- Кроме того существует ряд функций совпадающих по именам со структуральными типами (list,table,record), но они являются

Работа с файлами

- Для открытия файла необходимо использовать конструкцию:
  f := open(«путь до файла», «режим открытия»)
- Режимов открытия существует несколько и все они представлены в виде строк:

«r» - только чтение

«w» - только запись

«a» - дозапись файла (открытие файла и запись в его конец)

«c» - создать файл

- Возможно комбинировать режимы открытия.

Чтение из файла

- Для прочтения строки из файла можно использовать традиционный оператор !:

write(!f) # напечатает первую строку из файла

- Однако, есть специальные функции для чтения из файлового потока:read(f) и reads(f,n). Первая функция читает строку из файла, а вторая n символов из файла.
- Если n не указан, то он считается равным 1.

Нетрадиционное считывание файлов

- Из-за парадигмы целенаправленного выполнения следует несколько необычных возможностей Icon по считыванию данных.
- Например:
  while write(read(f)) # выведет весь файл на экран построчно
- Или еще один типовой ход:

```
while i := read(f) do {
   действия со строкой
}
```

Нетрадиционное считывание файлов

- Также допустимо использовать следующий ход:

every write(read(f))

- Или его сокращенный аналог:

every write(!f)

- Естественно, такой подход не ограничен одними подобными примитивами и позволяет создавать однострочную обработку данных из файла.

Запись в файл

- Запись в файл сделана достаточно тривиально и допускает использование тех же приемов, что и используемые при считывании.
- Для записи в файл используется конструкция:

write(«путь к файлу», expr)

где в роли expr может быть все что угодно, что преобразуемо в строку.

- Выражение writes(f,expr) запишет expr в файл посимвольно, но без перевода строки после записи

Закрытие файлового потока

- После работы с файлом нужно обязательно закрыть файловый поток во избежание затирания данных, порчи файла или же полного уничтожения данных, находившихся в файле.
- Для закрытия используется тривиальная функция close(f)
- Естественно, допускается использования файловых функций, типа открытие/закрытие в условиях и в условиях циклов.

Работа с ОС

- В Icon существует небольшой набор инструкций для общения с операционной системой:

system(cmd) — выполнение команды системы

remove(file) — удалить файл

rename(name1,name2) — переименовать файл

- Работа этих функций зависит от используемой ОС.

Сопоставление с образцом

- В некоторых случаях выражение генерирует больше двух возможных вариантов, исходя из которых программа должна принять решение.
- Как правило использование вложенных условий для уже рассчитанных значений весьма не очень удачный вариант.
- Удобнее в этом плане применить сопоставление значения выражения с некоторыми уже посчитанными значениями («образцами»)

Сопоставление с образцом

- Для этого существует оператор case:

```
case expr of {
  вариант 1 : действие 1
  вариант 2 : действие 2
  ...
}
```

- Оператор case вычисляет значение expr и сравнивает его с вариантами. В случае совпадения с одним из вариантов выполняется соответсвующее ему действие, иначе

Значения по умолчанию в case

- Иногда возникают ситуации когда необходимо принять решение даже в том случае, если значение выражения после case не сопало с образцом.
- Для того чтобы указать действия выполняющееся в этом случае необходимо применить следующую конструкцию:

```
case expr of {
  вариант 1 : действие 1
  вариант 2 : действие 2
  ...
```

Процедуры с переменным числом аргументов

- Рассмотрим построение процедур с переменным числом аргументов на примере линейной функции y = a + b * x
- Нетрудно заметить, что аргумент a нам нужен далеко не всегда (если его установить в ноль, то функция останется линейной). Этой особенностью мы воспользуем для построения универсальной линейной функции которая может принимать от 2 до 3 аргументов.

Процедуры с переменным числом аргументов

- Для построения этой функции достаточно воспользоваться операторами /, который покажет определена ли некоторая переменная:

```
procedure linear(a,b,x)
/a := 0 # если a не определена, то установить ее в ноль
return a + b * x
end
```

- Примеры использования:

```
linear(1,2,3) # равно 7
```

Процедуры с переменным числом аргументов

- Следующий хитроумный трюк позволит нам создать настоящие функции с переменным количеством аргументов: достаточно просто передавать функции один аргумент, который будет списком !
- Поскольку, список — структура динамичная, то можно только что созданный список протолкнуть на вход такой функции используя форму:

имя_процедуры ! Список

При этом сама процедура должна быть

Процедуры с переменным числом аргументов

- Например, можно показать как использовать список фиксированный длины и протолкнуть его в процедуру (не обязательно с переменным числом аргументов).
- Воспользуемся для примера функцией linear:

```
linear ! [1,2,3] # равно 7
```