ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК ПРОГРАММИРОВАНИЯ EL

Малявко Александр Антонович

к.т.н., доцент каф. Вычислительной техники НГТУ

ФУНКЦИОНАЛЬНОСТЬ : ИМПЕРАТИВНОСТЬ

- Императивность:
  - изменяемое состояние (переменные)
  - управляющие структуры, такие как циклы
- Функциональность:
  - неизменяемые переменные
  - рекурсия вместо циклов
  - функции первого и высшего порядков

ПРОТОТИП: ЯЗЫК ЭРЛАНГ

- Масса достоинств: акторный параллелизм, высокоуровневые структуры данных, горячая смена кода, простота отладки, ...
- Недостатки, обусловившие довольно узкие области применения:
  - Не чистые пользовательские функции, возможны побочные эффекты
  - Невозможность иметь хотя бы некоторые изменяемые переменные
  - Нет локализуемых строк, поддержка Unicode появилась недавно и неудобна
  - Странный синтаксис, затрудняющий рефакторинг
  - Медленная обработка числовых данных

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
ЛЕКСИКА

- Идентификаторы – самые обычные (*evtCnt*, *ind*)
- Ключевые слова – выделенные идентификаторы (*when*, *by*, *of*, *int* …)
- Числовые литералы – целые, вещественные, целые в системе счисления от 2 до 62 (*2, 3.1, 8$77*)
- Атомы – последовательности произвольных символов в одиночных апострофах (*'true'*, *'name'*)
- Строки - последовательности произвольных символов в двойных кавычках ("*string*")
- Знаки операций, скобки, … (=, +=, =:=, $^=, …)

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
СТРУКТУРА ПРОГРАММЫ

```
module blas;
import std, io;
/*comment /* module_initialyzer */ continue comment*/
macro koeff(arg) arg[#][##] / arg[##][##]
include recordsDef.elh
public g( m ) when ?(m) == 'vector' & ?(m[_]) == 'vector' &
        ?(m[_][_]) == 'number' & m.size()+1 == m[^].size() {
var m;
   m[_+1 .. ^] –= m[_ .. #-1] * koeff(m);
   m[^-1 .. _ ] –= m[^..#-1] * koeff(m);
   m[..][^]/m[#][#];          //return result vector
}
{nothing;}    // guard has a false meaning, return nothing
```

Функционально-императивный язык EL
Функции и данные

- В любой функции доступны только собственные локальные переменные и константы, определенные инициализаторами модуля
- Между функциями аргументы и возвращаемые результаты передаются только по значению
- Все функции – чистые, сбор мусора локален
- Переменные могут быть как иммутабельными, так и переприсваиваемыми:
  - по умолчанию любая необъявленная переменная иммутабельна, значение ей можно присвоить только один раз
  - переменную можно объявить как переприсваиваемую:

```
var eventCount, eventData;
...
eventCount += 1;
```

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
ТИПИЗАЦИЯ

- Типизация вариативна: статическая и/или динамическая
- Типы данных объявляются с использованием ключевых слов *int*, *long*, *bigint*, *float*, *double*, *bigfloat*, *string*, *atom*, *vector*, *list*, *tuple*, *binary*, *function*, *module*
- *Типы int, long, float, double* являются примитивными, остальные – объектными
- Типы данных можно объявлять в предложениях *var* или *def*

```
var float[numSensors] sumValues, currValue;
def tuple fieldNames; //кортеж, иммутабельная
```

- Все необъявленные переменные – объектные, в том числе – получившие численные значения

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
ТИПЫ ДАННЫХ

Здесь представлены только отличия от прототипа

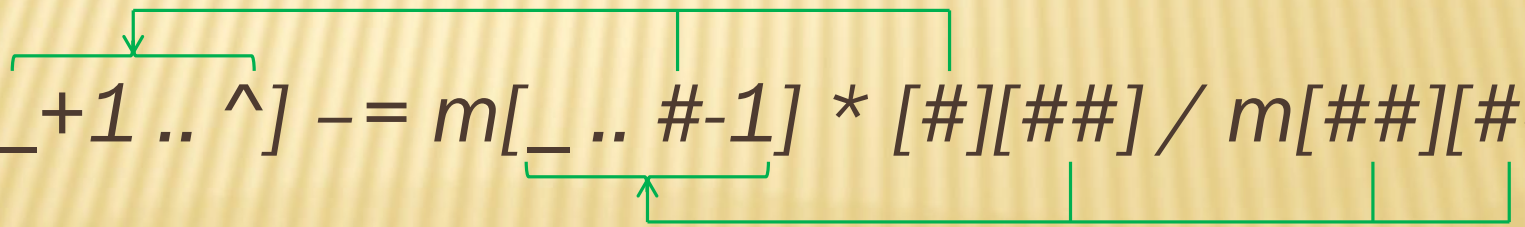
- Явные объявления переменных численных типов позволят компилятору построить более быстрый и компактный объектный код их обработки по сравнению с обработкой численных значений объектных переменных
- Списки в отличие от прототипа двусвязны и имеют две точки доступа: голову и хвост
- Соответственно расширены средства генерации списков (*list comprehension*)
- Вместо встроенных функций Эрланга для всех объектных переменных (и литералов) определены методы наподобие: *.size(), .hi(), ...*

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
НОВЫЙ (НЕТ В ЭРЛАНГЕ) ТИП ДАННЫХ - ВЕКТОР

- Добавлен (по сравнению с Эрлангом) тип данных *vector* как массив однотипных элементов
- Векторы требуют меньше памяти по сравнению со списками однотипных элементов
- Векторы могут быть как прямоугольными, так и ступенчатыми
- Обработка прямоугольных векторов быстрее
- Операции с векторами примитивных типов могут быть оптимизированы по времени
- К векторам в целом могут применяться многие операции (+, - ...), это также предоставляет большую свободу компилятору

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
ОСОБЕННОСТИ РАБОТЫ С ВЕКТОРАМИ

- Специальные обозначения нижней «_» и верхней «^» границ изменения индекса
- Встроенные методы *.lo()* – *возвращает* «_», *.hi()* – возвращает «^», *.size()* – возвращает разность «^» – «_»
- Возможность задания диапазонов изменения границ индексов, например: *m[_ .. ^-1]* или *matrix[^ .. _ step 2]*
- Индексные ссылки:



(используются текущие значения индексов)

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
ОПЕРАТОРЫ

- Пустой оператор: ;
- Выражение: строится из термов, знаков операций и круглых скобок (при необходимости), завершается ; Знаки операций:
  - присваивание/сопоставление с образцом «=»
  - присваивания с вычислениями «=», «+=», «$^=» ...
  - логические «&», «|», «~»
  - битовые «$&», «$|», «$~», «$^»
  - сравнения «>», «>=», «<», «<=», «>», «<>»,«==», «===» ...
  - арифметические «+», «-», «*», «/», «\», «%», «++», «--»
  - атомарная перестановка «=:=»
  - обращения к элементам и методам объектов «.», «$»
  - ...
- Возврата из функции: *return* <*expression*> ;

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
ОПЕРАТОРЫ

- Оператор ветвления. Форма 1:

```
by <::метка>@ <выражение> {
   of <выражение> : <послед._операторов>
   of ...        // другие ветки
}
```

- Оператор ветвления. Форма 2:

```
by <::метка>@ <выражение>@ {
   when <лог.выражение> : <послед._операторов>
   when ...  // другие ветки
   else : <послед._операторов> //необязательно
}
```

- break, break <метка>, again, again <метка>, next

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
ОПЕРАТОРЫ

```
public bubbleSort(arr) when ?(arr) == 'vector' & ?(arr[ _ ]) == 'number' {
var arr, top, ind;
    by::outer top = arr.^ { //внешний цикл, поднимающий пузырек до границы
        when top-- > (ind = arr._) : //продолжается, пока уменьшающийся
    // индекс (top--)текущей границы не дойдет до начала массива
            by::inner {  //внутренний цикл, сравнивающий пару элементов
                when ind++ <= top : //если не вышли за верхнюю границу
                    by arr[ind - 1] > arr[ind] {  //то сравниваем соседние элементы
                        of true : arr[ind - 1] =:= arr[ind]; //и переставляем, если нужно
                    }
                    again inner;        //на повтор внутреннего цикла
            }
            again outer;  //на повтор внешнего цикла
    }
    arr;  //возвращаем из функции отсортированный массив
}
{ arr; } //последний неохраняемый блок возвращает неизмененный аргумент
```

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
ОПЕРАТОРЫ

- Блок операторов:

{<объявления переменных>@
<последовательность операторов>}

- Оператор перехвата исключений:

try <block>

catch {

of <выражение> : <послед._ операторов>

of …

}

finally <block>

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
СВОЙСТВА И ХАРАКТЕРИСТИКИ

- Высокоуровневые типы данных и операций над данными. В том числе – операции над векторами в целом, упрощающие распараллеливание их исполнения компилятором
- Отсутствие сборщика мусора и в то же время – явного захвата и освобождения памяти
- Доступны функции высшего порядка, анонимные функции, замыкания, т.е. общеизвестные достоинства функциональной парадигмы
- Реализован полиморфизм на уровне сигнатур, а не имен функций
- Возможность использовать статическую типизацию с целью организации эффективной обработки данных примитивных типов

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
СВОЙСТВА И ХАРАКТЕРИСТИКИ

- Реализована возможность создавать и использовать объекты, но нет ни интерфейсов, ни наследования
- Немногословность, простота и удобство управляющих конструкций
- Имеется возможность явного задания циклических вычислений вместо более дорогостоящих рекурсивных вызовов, которые, впрочем, тоже можно использовать
- Ожидаемая простота тестирования и отладки, обусловленная тем, что нет глобальных переменных и указателей, нет побочного эффекта у функций, все функции чистые.

ФУНКЦИОНАЛЬНО-ИМПЕРАТИВНЫЙ ЯЗЫК EL
СОСТОЯНИЕ РЕАЛИЗАЦИИ

- Полностью разработаны (на C++) и отлажены лексический и синтаксический анализаторы, реализована глубокая нейтрализация синтаксических ошибок.
- Частично (для нескольких типов данных – числа, строки, списки и еще не для всех операторов) реализован преобразователь в псевдокод и далее – на язык ассемблера LLVM.
- Параллельно ведется работа по написанию компилятора для EI на языке EI.

БЛАГОДАРЮ ЗА ВНИМАНИЕ!

ГОТОВ ОТВЕЧАТЬ НА ВОПРОСЫ