Простой Рефал. Руководство пользователя

Оглавление

[1 Язык Простой Рефал: синтаксис и семантика 1](#_Toc527730223)

[1.1 Программные элементы (объявления и определения) 1](#_Toc527730224)

[1.1.1 Объявления функций ($FORWARD и $EXTERN) 1](#_Toc527730225)

[1.1.2 Регулярные функции 2](#_Toc527730226)

[1.2 Функции, семантика и сравнение с РЕФАЛом-5 2](#_Toc527730227)

[1.2.1 Синтаксис функций, общее с РЕФАЛом-5 2](#_Toc527730228)

[2 Библиотека функций 2](#_Toc527730229)

[2.1 Расширенная библиотека (LibraryEx) 3](#_Toc527730230)

[2.2 LibraryEx: надстройки над функциями базовой библиотеки и другие полезные функции 3](#_Toc527730231)

[2.2.1 LoadFile 3](#_Toc527730232)

[2.2.2 SaveFile 3](#_Toc527730233)

[2.2.3 Inc, Dec 3](#_Toc527730234)

[2.2.4 ArgList 3](#_Toc527730235)

[2.2.5 Trim 3](#_Toc527730236)

[2.3 LibraryEx: функции высших порядков 3](#_Toc527730237)

[2.3.1 Понятие функтора, Apply 3](#_Toc527730238)

[2.3.2 Map 5](#_Toc527730239)

[2.3.3 Reduce 5](#_Toc527730240)

[2.3.4 MapReduce 5](#_Toc527730241)

[2.3.5 DelAccumulator 6](#_Toc527730242)

[3 Интерфейс с языком C++ 6](#_Toc527730243)

[3.1 Вычислительная модель 6](#_Toc527730244)

[3.2 Написание внешних функций 6](#_Toc527730245)

[3.2.1 Быстрый и грязный способ 6](#_Toc527730246)

[3.2.2 Написание функции вручную 6](#_Toc527730247)

[4 Список литературы 6](#_Toc527730248)

# Язык Простой Рефал: синтаксис и семантика

## Программные элементы (объявления и определения)

### Объявления функций ($FORWARD и $EXTERN)

…

Объявления функций на Рефале компилируются в соответствующие объявления функций на C++, $FORWARD — с модификатором static, $EXTERN — с модификатором extern.

Пример:

$FORWARD A, B, C;

$EXTERN E, F, G;

Компилируется в:

static refalrts::FnResult A(refalrts::Iter arg\_begin, refalrts::Iter arg\_end);

static refalrts::FnResult B(refalrts::Iter arg\_begin, refalrts::Iter arg\_end);

static refalrts::FnResult C(refalrts::Iter arg\_begin, refalrts::Iter arg\_end);

extern refalrts::FnResult E(refalrts::Iter arg\_begin, refalrts::Iter arg\_end);

extern refalrts::FnResult F(refalrts::Iter arg\_begin, refalrts::Iter arg\_end);

extern refalrts::FnResult G(refalrts::Iter arg\_begin, refalrts::Iter arg\_end);

### Регулярные функции

…

Все разновидности функций (пустые функции, статические ящики, регулярные функции) компилируются в функции C++ со следующей сигнатурой:

static refalrts::FnResult *ЛокальнаяФункция*(  
 refalrts::Iter arg\_begin, refalrts::Iter arg\_end  
) {

…

}

refalrts::FnResult *EntryФункция*(refalrts::Iter arg\_begin, refalrts::Iter arg\_end) {

…

}

Смысл возвращаемого значения и параметров arg\_begin и arg\_end будет рассмотрен в разделе, посвящённом интерфейсу с языком C++.

## Функции, семантика и сравнение с РЕФАЛом-5

…

### Синтаксис функций, общее с РЕФАЛом-5

Общие черты языков Простой Рефал и РЕФАЛ-5 (подмножество Базисного РЕФАЛа):

* …
* *(Относится не к языку, а к реализации)* Затраты времени на операции. Реализация Простого Рефала использует классическое списковое представление, поэтому особенности стоимости отдельных операций (сопоставление с открытыми и повторными переменными, стоимость конкатенации и копирования) те же, что и в реализации РЕФАЛа-5 (версия PZ Oct 29 2004).

…

# Библиотека функций

В дистрибутив Простого Рефала входит стандартная библиотека языка, представленная двумя единицами трансляции — Library.cpp (далее на неё будем ссылаться как Library) и LibraryEx.sref (далее — LibraryEx). Первая содержит набор первичных функций, т. е. функций, которые невозможно выразить на Рефале: преобразования атомов, арифметические функции, ввод-вывод, вторая — набор удобных функций, написанных на Рефале, и использующих библиотеку Library. Прежде чем переходить к рассмотрению функций, входящих в библиотеку, сначала введём нотацию для описания форматов функций (про форматы функций см. (Турчин, 1989)).

## Расширенная библиотека (LibraryEx)

Эта библиотека содержит набор функций, написанных на Рефале и предназначенных для упрощения процесса программирования. В неё входит ряд функций высшего порядка (типа Map или Reduce), а также удобные обёртки над функциями библиотеки Library (такие как Inc или LoadFile).

Для пояснения семантики ряда функций помимо формата будет приводиться и исходный код самих функций.

## LibraryEx: надстройки над функциями базовой библиотеки и другие полезные функции

### LoadFile

<LoadFile e.FileName> == (e.Line)\*

Загружает файл в поле зрения, каждая строка файла заворачивается в отдельный скобочный терм. Если файл заканчивается на пустую строку, она игнорируется. Для чтения файла используется функция FReadLine.

### SaveFile

<SaveFile (e.FileName) (e.Line)\*> == пусто

Сохраняет последовательность строк (e.Line)\* в файле с заданным именем. Для записи строк используется функция FWriteLine.

### Inc, Dec

<Inc s.Number> == s.Number  
<Dec s.Number> == s.Number

$ENTRY Inc {

s.Num = <Add s.Num 1>;

}

$ENTRY Dec {

s.Num = <Sub s.Num 1>;

}

Функции увеличивают и уменьшают число на единицу. Запись <Inc s.Number> компактнее и выразительнее, чем <Add s.Number 1>.

### ArgList

<ArgList> == (e.Arg)+

Возвращает аргументы командной строки <Arg 0> <Arg 1> … до первого пустого значения, возвращённого функцией Arg. Если среди аргументов программы присутствует пустой аргумент, аргументы после него будут проигнорированы.

### Trim

<Trim e.String> == e.String’

Функция удаляет пробельные символы (пробелы, табуляции, переводы строк CR и LF) в начале и в конце строки.

## LibraryEx: функции высших порядков

### Понятие функтора, Apply

Большинство функций высших порядков, представленных в библиотеке, оперирует таким объектом, как функтор:

t.Functor ::= s.Closure | (t.Functor e.BoundedArgs)

Функтор может быть либо экземпляром функции, либо скобочным термом, первым элементом которого является функтор. Вызов функтора осуществляется функцией Apply, которая разворачивает скобочные термы до тех пор, пока первым термом не будет атом, который затем и вызывает:

<Apply t.Function e.Args> ≈≈ <t.Function e.Args>

t.Function ::=

s.Closure | (t.Function e.BoundedArgs)

$ENTRY Apply {

s.Fn e.Argument = <s.Fn e.Argument>;

(t.Closure e.Bounded) e.Argument =

<Apply t.Closure e.Bounded e.Argument>;

}

Функции высших порядков (кроме функции Y), определённые ниже, используют функцию Apply для вызова функторов.

Исторически функторы использовались для имитации функциональности замыкания с контекстом: писалась глобальная функция, указатель на неё «связывался» с некоторым значением, которое играло роль контекста. Функция CardProd, рассмотренная ранее, с использованием глобальных функций и функторов выглядела бы так:

// Функция CardProd вычисляет декартово произведение двух множеств  
// <CartProd ('a' 'b' 'c') (1 2)>[[1]](#footnote-1)  
// == ('a' 1) ('a' 2) ('b' 1) ('b' 2) ('c' 1) ('c' 2)

$FORWARD CartProd-By-tA, CartProd-By-tB;

$ENTRY CartProd {

(e.SetA) (e.SetB) =

<Map (CardProd-By-tA e.SetB) e.SetA>;

}

CartProd-By-tA {

e.SetB t.A =

<Map (CartProd-By-tB t.A) e.SetB>;

}

CartProd-By-tB {

t.A t.B = (t.A t.B);

}

Как правило, удобнее использовать вложенные функции вместо функторов, однако функторы могут оказаться полезными в тех случаях, когда действие сводится исключительно к вызову *имеющейся* функции с фиксированной частью аргумента. Например, вызов

<Map (Add 3) e.Numbers>

лаконичнее и читабельнее, чем

<Map { s.Number = <Add 3 s.Number>; } e.Numbers>

Другой пример будет приведён в примере с функцией Fetch.

### Map

<Map t.Func t.Elem\*> == <t.Func t.Elem>\*

$ENTRY Map {  
 t.Fn t.Next e.Tail = <Apply t.Fn t.Next> <Map t.Fn e.Tail>;  
  
 t.Fn = ;  
}

Функция Map последовательно применяет функтор к каждому терму своего аргумента.

### Reduce

<Reduce t.Func t.Acc t.Elem\*> == t.Acc”,  
<t.Func t.Acc t.Elem> == t.Acc’

$ENTRY Reduce {  
 t.Fn t.Acc t.Next e.Tail =  
 <Reduce  
 t.Fn <Apply t.Fn t.Acc t.Next> e.Tail  
 >;  
  
 t.Fn t.Acc = t.Acc;  
}

Функция Reduce сворачивает каждый элемент последовательности (в порядке слева-направо) с термом-аккумулятором с применением данного функтора. Пример:

<Reduce Add 0 e.Numbers> // вычисляет сумму чисел в e.Numbers  
<Reduce Max 0 e.Numbers> // вычисляет максимальное число  
 // (определение Max см. в примере к Fetch)

### MapReduce

<MapReduce t.Func t.Acc t.Elem\*> == t.Acc” e.Mapped\*,  
где <t.Func t.Acc t.Elem> == t.Acc’ e.Mapped

Функция MapReduce сочетает в себе свойства функций Map (преобразование каждого элемента последовательности функтором) и Reduce (свёртка каждого элемента последовательности с аккумулятором). Функтор принимает два терма: текущее значение аккумулятора и очередной элемент, возвращает модифицированное значение аккумулятора и некоторое объектное выражение — результат трансформации элемента. Функция MapReduce применяет функтор последовательно к каждому элементу и очередному значению аккумулятора, возвращает результирующее значение аккумулятора и результаты трансформации всех элементов. Приведём пример — напишем функцию, которая загружает файл и нумерует в нём строчки:

LoadNumerated {  
 e.FileName =  
 <DelAccumulator // См. далее  
 <MapReduce  
 {  
 s.Number (e.Line) =  
 <Inc s.Number> (s.Number e.Line);  
 }  
 1 // начальное значение счётчика  
 <LoadFile e.FileName>  
 >  
 >;  
}

Вложенная функция принимает два терма: номер текущей строки и строку из файла, возвращает номер следующей строки (на единицу больше) и преобразованную строку файла, первым термом содержащую число — её номер. Значение аккумулятора после анализа нам не нужно, оно отбрасывается при помощи функции DelAccumulator.

### DelAccumulator

<DelAccumulator t.Acc e.Data> == e.Data

$ENTRY DelAccumulator {  
 t.Acc e.Tail = e.Tail;  
}

Функция DelAccumulator отбрасывает первый терм от своего аргумента, используется в сочетании с функцией MapReduce — когда значение аккумулятора по завершении вычислений стало уже не нужным.

# Интерфейс с языком C++

## Вычислительная модель

## Написание внешних функций

### Быстрый и грязный способ

### Написание функции вручную

# Список литературы

Алешин, А. Ю., Красовский, А. Г., Романенко, С. А., & Шерстнев, В. Ю. (1991). *Система программирования РЕФАЛ-2 для IBM PC, PDP-11 и VAX-11. Руководство пользователя.* Москва.

Немытых, А. П. (2014). Лекции по языку программирования РЕФАЛ. В *Cборник трудов по функциональному языку программирования Рефал* (Т. 1, стр. 118-165). Переславль-Залесский: Изд-во "Сборник". Получено из http://refal.botik.ru/library/refal2014\_issue-I.pdf

Скоробогатов, С. Ю., & Чеповский, А. М. (2006). Язык Refal с функциями высших порядков. *Информационные технологии*(№ 9). Получено из http://iu9.bmstu.ru/science/refal.pdf

Турчин, В. Ф. (1989). *Фрейм: РЕФАЛ-5. Руководство и справочник.* Получено 16 апреля 2015 г., из Содружество «РЕФАЛ/Суперкомпиляция»: http://refal.ru/rf5\_frm.htm

1. Нотация, используемая в комментарии, объясняется в разделе 4.1. [↑](#footnote-ref-1)