

Федеральное агентство по образованию РФ
Пермский Национальный Исследовательский Политехнический Университет
Кафедра Информационных Технологий и Автоматизированных Систем

Кузнецов Д.Б., Вагин Д.А.

Функциональное и логическое программирование

Методические указания по выполнению лабораторных работ
для специальностей АСУ и ПОБТ

г. Пермь 2011

1 Введение

Предлагаемый лабораторный практикум предназначен для освоения основных принципов программирования на функциональных языках программирования LISP и Haskell, и логическом языке программирования Prolog.

1.1 Требования к аппаратному обеспечению

Вариант 1: Выполнение работ на локальном компьютере Компьютер должен обеспечивать возможность запуска операционной системы, текстового редактора, и средств компиляции исходного кода. Необходимо иметь 200МБ свободного места на жёстком диске для установки необходимого ПО и хранения исходных текстов программ.

Вариант 2: Выполнение работ на удалённом сервере Сервер должен иметь технические характеристики достаточные для запуска операционной системы и сервисов для удалённого подключения. Более точные аппаратные характеристики зависят от версии ядра и системных библиотек. Для запуска последних версий без графического режима вполне достаточно процессор с частотой от 300 МГц, ОЗУ от 512 МБ, свободно дисковое пространство от 500 МБ. Локальное рабочее место должно быть оборудовано терминалом, подключенным к серверу.

1.2 Требования к системному программному обеспечению

Рассматриваемое ниже системное программное обеспечение должно быть установлено на локальном компьютере (Вариант 1 аппаратного обеспечения) или на сервере (Вариант 2 аппаратного обеспечения).

Для выполнения лабораторных работ потребуется инструментарий для сборки компиляции программ на C или C++ (GNU make, GNU gcc, Microsoft Visual Studio, и т.д.), интерпретатор языка LISP (clisp, autolisp, и т.д.), компилятор языка Haskell (ghc), интерпретатор языка Prolog (gprolog).

2 Лабораторная работа №1.

Сравнение циклов и рекурсии

2.1 Цели

- Оценить недостатки процедурного программирования
- Научиться строить рекурсивные алгоритмы

2.2 Порядок выполнения

- 1 Написать программу по заданию с использованием цикла
- 2 Провести трассировку программы
- 3 Составить рекурсивную функцию для решения выданного задания
- 4 Реализовать составленную рекурсивную функцию на языке программирования
- 5 Написать отчет

2.3 Рекомендации по выполнению

- Массивы фиксированной длины
- Трассировка отключается макросом
- Данные задаются внутри исходного кода

2.4 Состав отчета

- Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
- Текст рекурсивной функции
- Текст итеративной функции
- Результаты выполнения

2.5 Варианты заданий

- 1 Напишите программу печатающую n -ое число Фибоначчи.
- 2 Напишите программу вычисляющую факториал натурального числа.
- 3 Напишите программу перемножающую два целых неотрицательных числа без использования операции умножения.
- 4 Напишите программу, печатающую значение многочлена степени $n \geq 0$ в заданной точке x_0 . Коэффициенты многочлена хранятся в массиве a в порядке убывания степеней и являются целыми числами, так же как и значение x_0 .
- 5 Напишите программу печатающую значение производной многочлена степени $n \geq 0$ в заданной точке x_0 . Коэффициенты многочлена хранятся в массиве a в порядке убывания степеней и являются целыми числами, так же как и значение x_0 .
- 6 Напишите программу возводящую целое число в целую неотрицательную степень.

- 7 Напишите программу принимающую на вход натуральное число и выводящую Yes если число является простым, и No - если не является.
- 8 Напишите программу генерации всех правильных скобочных структур длины $2n$. Например для $n = 3$ таких структур может быть 5: $()()()$, $((()))$, $()(())$, $((()))$, $((()))$.
- 9 Имеется три стержня А, В, С. На стержень А нанизано n дисков радиуса $1, 2, \dots, n$ таким образом, что диск радиуса i является i -м сверху. Требуется переместить все диски на стержень В, сохраняя их порядок расположения (диск с большим радиусом находится ниже). За один раз можно перемещать только один диск с любого стержня на любой другой стержень. При этом должно выполняться следующее условие: на каждом стержне ни в какой момент времени никакой диск не может находиться выше диска с меньшим радиусом.
- 10 Напишите программу выводящую сумму квадратов всех натуральных чисел от 1 до введённого n .
- 11 Напишите программу печатающую n -ое простое число.
- 12 Напишите программу, печатающую старшую цифру в десятичной записи введенного натурального числа.
- 13 Напишите программу, печатающую количество цифр в десятичной записи введенного натурального числа.
- 14 Напишите программу, печатающую количество натуральных решений неравенства $x^2 + y^2 < n$ для введенного n .
- 15 Напишите программу, вводящую натуральное число, и печатающую количество точек с целочисленными координатами внутри замкнутого шара радиуса с центром в начале координат.
- 16 Напишите программу, печатающую квадраты всех целых чисел от нуля до введенного натурального n , не использующую операций умножения.
- 17 Напишите программу, находящую количество счастливых билетов с шестизначными номерами. Билет называется счастливым, если сумма его первых трех цифр равна сумме трех последних.

2.6 Пример

Задание: Напишите программу проверяющую является ли введённое число факториалом какого либо числа.

2.6.1 Итеративное решение

Возьмём математическое определение факториала:

$$n! = 1 \cdot 2 \cdot \dots \cdot n = \prod_{i=1}^n i \quad (1)$$

Получается что факториал числа n должен делиться нацело на все натуральные числа до n включая n . Напишем программу реализующую такую проверку:

```

#include <stdio.h>

int check_factorial_iterate(int number){
    if (number < 0)
        return 0;
    if (number == 0)
        return 1;
    int i = 1;
    int n = 1;
    for (; n<number; n*=i, i++){
        if (number%i != 0)
            return 0;
    }
    return 1;
}

int main(){
    if (check_factorial_iterate(362880))
        printf("%s", "yes");
    else
        printf("%s", "no");
    return 0;
}

```

Listing 1: итеративная программа

2.6.2 Рекурсивное решение

Возьмём рекурсивное определение факториала:

$$n! = \begin{cases} 1 & n = 0, \\ n \cdot (n-1)! & n > 0. \end{cases} \quad (2)$$

```

#include <stdio.h>

int check_factorial_recursive(int number, int i){
    if (number < 0)
        return 0;
    if (number == 0)
        return 1;

    if (number == 1)
        return 1;

    if (number%i != 0)
        return 0;
    else
        return check_factorial_recursive(number/i, i+1);
}

int main(){
    int number = 362881;

    if (check_factorial_recursive(number, 1))
        printf("%s\n", "yes");
    else
        printf("%s\n", "no");

    return 0;
}

```

}

Listing 2: рекурсивная программа

3 Лабораторная работа №2

S-выражения в LISP

3.1 Цели

- Освоить S-выражения
- Научиться основам работы в clisp
- Познакомиться с функциями обработки списков

3.2 Задание

- 1 Составить список по заданию в синтаксисе lisp
- 2 Написать функции для получения каждого из элементов списка №1
- 3 Написать функцию для получения списка №2

3.3 Пример

Задание

- 1 Список №1 - $\{\{1, \{2, 3, 4, 5, 6, 7, 8\}\}, 9\}$
- 2 Список №2 - $\{2, 8, 3, \{4, 1\}, 6\}$

```
'((1 (2 3 4 5 6 7 8)) 9)
```

Listing 3: Задание 1

```
(car '((1 (2 3 4 5 6 7 8)) 9))  
;; (1 (2 3 4 5 6 7 8))  
(car (car '((1 (2 3 4 5 6 7 8)) 9)))  
;; 1  
(car (cdr (car '((1 (2 3 4 5 6 7 8)) 9))))  
;; (2 3 4 5 6 7 8)  
(car (car (cdr (car '((1 (2 3 4 5 6 7 8)) 9)))))  
;; 2  
(setq a '((1 (2 3 4 5 6 7 8)) 9))  
;; ((1 (2 3 4 5 6 7 8)) 9)  
(car (car (cdr (car a))))  
;; 2  
(car (cdr (car (cdr (car a)))))  
;; 3  
(car (cdr (cdr (car (cdr (car a))))))  
;; ...
```

Listing 4: Задание 2

```
(cons 2 (cons 8 (cons 3 (cons (cons 4 (cons 1 nil)) (cons 6 nil)))))  
;; (2 8 3 (4 1) 6)
```

Listing 5: Задание 3

3.4 Состав отчета

- Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
- Текст рекурсивной функции
- Результаты выполнения

3.5 Варианты заданий

- Список №1 - $\{1, \{2, 3, 4\}, 5, \{6, \{7, 8\}, 9\}\}$
 - Список №2 - $\{2, 8, \{3, 4, 1\}, 6\}$
- Список №1 - $\{\{1, 2, \{3, 4, 5, 6\}, 7\}, 8, 9\}$
 - Список №2 - $\{2, \{8, 3, 4, 1\}, 6\}$
- Список №1 - $\{\{\{1, 2, 3, 4\}, 5, 6, 7, 8\}, 9\}$
 - Список №2 - $\{\{7, 8\}, \{3, 4\}, \{1, 6\}\}$
- Список №1 - $\{1, 2, \{3, 4, 5\}, \{\{6, 7, 8, 9\}\}\}$
 - Список №2 - $\{\{2, 8\}, 6, 7, 8, 9\}$
- Список №1 - $\{1, 2, \{3, 4, 5\}, \{6, 7, \{8, 9\}\}\}$
 - Список №2 - $\{2, 8, \{3, 4, 5\}, 6\}$
- Список №1 - $\{1, \{\{2, 3, 4\}, 5, 6, 7\}, 8, 9\}$
 - Список №2 - $\{2, \{\{8\}, 3, 4, 1, 6\}\}$
- Список №1 - $\{1, \{2, 3, \{4, 5, 6\}, 7, 8\}, 9\}$
 - Список №2 - $\{2, \{8, 3, 4, \{1\}\}, 6\}$
- Список №1 - $\{\{\{1, 2, 3, 4\}, 5, 6, 7, 8\}, 9\}$
 - Список №2 - $\{2, 8, 3, \{4, 1\}, 6\}$
- Список №1 - $\{\{\{1, 2, 3, 4, 5, 6, 7, 8\}, 9\}\}$
 - Список №2 - $\{2, 8, 3, 4, \{1, 6\}\}$
- Список №1 - $\{1, 2, 3, \{4, \{5, 6, \{7, 8, 9\}\}\}\}$
 - Список №2 - $\{2, \{8, 3\}, 4, 1, 6\}$
- Список №1 - $\{1, \{2, 3, \{4, 5, 6, \{7, 8\}, 9\}\}\}$
 - Список №2 - $\{2, 8, 3, 4, \{\{1\}, 6\}\}$
- Список №1 - $\{1, \{\{2, 3\}, 4, 5, 6\}, 7, 8, 9\}$
 - Список №2 - $\{\{2\}, 8, \{3, \{4, 1\}\}, 6\}$
- Список №1 - $\{1, \{2, \{3\}, 4, 5\}, 6, \{7, 8\}, 9\}$
 - Список №2 - $\{2, \{\{8, 3\}, \{4, 1\}\}, 6\}$
- Список №1 - $\{\{1, \{2\}, 3, 4\}, 5, 6, \{7, \{8, 9\}\}\}$
 - Список №2 - $\{2, \{8, 3, \{4\}, 1\}, \{6\}\}$
- Список №1 - $\{1, \{2, 3, \{4, 5, \{6, \{7\}\}\}, 8\}, 9\}$

- Список №2 - $\{2, \{8, \{3, \{4\}\}, 1\}, 6\}$
- 16
- Список №1 - $\{\{1\}, 2, \{3, 4\}, \{5, 6, \{7, 8\}, 9\}\}$
 - Список №2 - $\{\{2, 8\}, \{3, \{4, 1\}, 6\}\}$
- 17
- Список №1 - $\{1, \{2, 3\}, 4, \{5\}, 6, 7, 8, 9\}$
 - Список №2 - $\{2, \{8\}, \{3, 4, \{1\}\}, 6\}$
- 18
- Список №1 - $\{1, 2, \{3\}, \{4\}, \{5\}, 6, \{7, \{8\}\}, 9\}$
 - Список №2 - $\{\{2, \{8, 3\}, 4, 1\}, \{6\}\}$
- 19
- Список №1 - $\{1, 2, \{3, \{4\}, 5\}, \{6, \{7\}, 8\}, 9\}$
 - Список №2 - $\{\{2, \{8, 3, 4, \{1\}\}\}, 6\}$
- 20
- Список №1 - $\{\{1, 2\}, \{\{3, \{4\}, 5\}, \{6, \{7\}, 8\}\}, \{9\}\}$
 - Список №2 - $\{2, \{\{8\}, 3, \{4, 1\}\}, \{6\}\}$

4 Лабораторная работа №3

Функции в LISP

4.1 Цели

- Познакомиться именованными функциями
- Познакомиться с анонимными функциями

4.2 Задание

- 1 Написать функцию по первому заданию
- 2 Написать функцию принимающую в качестве аргумента список заданного вида и возвращающую список такого же вида, но с изменёнными значениями.
- 3 Написать анонимную функцию, которая передаётся параметром в функцию из второго задания, и выполняет некоторые действия над элементами списка.

4.3 Пример

Задание

- 1 Функция принимает два аргумента и возвращает их сумму.
- 2 Функция принимает список вида (x x x x x ...) и увеличивает каждый элемент списка на 1.
- 3 Написать анонимную функцию которая преобразует каждый элемент в список.
((x) (x) (x)...)

```
(defun summa (a b)
  (+ a b)
)
```

```
;; пример
(summa 4 5)
```

Listing 6: Задание 1

```
(defun mapx (x)
  (if x
      (cons (+ (car x) 1) (mapx (cdr x)))
      nil
  )
)
;; примервызова
(mapx '(1 2 3 4))
```

Listing 7: Задание 2

```
(defun mapx (x f)
  (if x
      (cons (funcall f (car x)) (mapx (cdr x) f))
      nil
  )
)
;; примервызова
(mapx '(1 2 3 4) (lambda (x)(+ x 1)))
```

Listing 8: Задание 3

4.4 Состав отчета

- Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
- Текст рекурсивной функции
- Результаты выполнения

4.5 Варианты заданий

- Функция принимает два числа, и если их сумма чётна, то возвращает их разницу, иначе - сумму.
 - Дан список $((x\ x)\ (x\ x)\ (x\ x)\ \dots)$. Увеличить каждый элемент на единицу.
 - Объединить каждый подсписок суммированием: $((1\ 2)\ (1\ 3)\ (3\ 4)) \rightarrow ((3)\ (4)\ (7))$
- Функция принимает два числа и возвращает наибольшее из них
 - Дан список $((x\ x\ x\ x\ \dots)\ (x\ x\ x\ x\ \dots))$. Увеличить каждый элемент на единицу
 - Умножить каждый элемент на произвольное число
- Функция принимает 3 числа и возвращает список с этими числами
 - Дан список $(x\ (x\ (x\ (x\ \dots))))$. Увеличить каждый элемент на единицу
 - Умножить каждый элемент на 2
- Функция принимает 1 число и возвращает квадрат этого числа если оно чётное, и куб, если нечётное
 - Дан список $((((\dots\ x)\ x)\ x)\ x)$. Увеличить каждый элемент на единицу
 - Поделить каждый элемент на 2
- Функция принимает 2 числа и возвращает их произведение, если их сумма чётна, и квадрат первого, если сумма нечётна
 - Дан список $((x\ (x\ (x)))\ (x\ (x\ (x)))\ (x\ (x\ (x)))\ \dots)$. Увеличить каждый элемент на единицу
 - Увеличить каждый элемент в 2 раза
- Функция принимает список и число, и добавляет число к списку.
 - Дан список $((x\ x\ x)\ (x\ x\ x)\ \dots)$. Увеличить каждый элемент на единицу.
 - Объединить каждый подсписок суммированием:
 $((1\ 2\ 1)\ (1\ 3\ 2)\ (3\ 4\ 1)) \rightarrow ((4)\ (6)\ (8))$
- Функция принимает список 3 числа и возвращает список вида $(x\ (x\ (x)))$
 - Дан список $((x)\ (x)\ \dots)$. Увеличить каждый элемент на единицу.
 - Преобразовать список в простой список элементов: $((1)\ (3)\ (4)) \rightarrow (1\ 3\ 4)$
- Функция принимает список 3 числа и возвращает список вида $((x\ x)\ x)$
 - Дан список $((x\ (x\ x))\ (x\ (x\ x))\ \dots)$. Увеличить каждый элемент на единицу.
 - Увеличить каждый элемент в произвольное число раз
- Функция принимает список 3 числа и возвращает список вида $((x)\ x\ (x))$
 - Дан список $((x\ x\ (x))\ ((x)\ x\ (x))\ \dots)$. Увеличить каждый элемент на единицу.
 - Увеличить каждый элемент в произвольное число раз

- 10
 - Функция принимает список 3 числа и возвращает список вида $(x (x) x)$
 - Дан список $((x) (x)) ((x) (x)) \dots$. Увеличить каждый элемент на единицу.
 - Увеличить каждый элемент в произвольное число раз
- 11
 - Функция принимает список 3 числа и возвращает список вида $((x) x (x))$
 - Дан список $((x) (x) \dots)$. Увеличить каждый элемент на единицу.
 - Увеличить каждый элемент в произвольное число раз
- 12
 - Функция принимает два числа и возвращает наименьшее из них
 - Дан список $((((\dots x) x) x) x)$. Увеличить каждый элемент на 2.
 - Увеличить каждый элемент в произвольное число раз
- 13
 - Функция принимает список и число, и добавляет число к списку с конца.
 - Дан список $((x) (x) \dots)$. Увеличить каждый элемент на единицу.
 - Поделить каждый элемент на 2
- 14
 - Функция принимает 3 числа и возвращает список с этими числами
 - Дан список $((x) (x) \dots)$. Увеличить каждый элемент на единицу.
 - Умножить каждый чётный элемент на произвольное число
- 15
 - Функция принимает два числа и возвращает первое, если оно кратно второму, и второе, если не кратно.
 - Дан список $((x x x x \dots) (x x x x \dots))$. Увеличить каждый элемент на единицу
 - Объединить каждый подсписок суммированием:
 $((1 2 1\dots) (1 3 2 \dots) (3 4 1 \dots)) \rightarrow ((4) (6) (8))$

5 Лабораторная работа №4

λ -выражения и β -редукция в λ -исчислении в языке LISP

5.1 Цели

- Изучить λ -исчисление
- Изучить β -редукцию в λ -исчислении

6 Задание

- 1 Выполнить β -редукции несколькими способами
- 2 Составить программу на LISP для вычисления функции

6.1 Пример

Задание

$$(((\lambda xyz.x(yz))(\lambda x.x \cdot 1))(\lambda x.x + x))2)$$

```
(  
  (lambda (x y z) (funcall x (funcall y z)))  
  (lambda (x) (* x 1))  
  (lambda (x) (+ x x))  
  2)
```

```
;; результат  
4
```

Listing 9: Программа на LISP

$$\begin{aligned} &(((\lambda xyz.x(yz))(\lambda x.x))(\lambda x.x + x))2 = \\ &= ((\lambda yz.(\lambda x.x)(yz))(\lambda x.x + x))2 = \\ &= (\lambda z.(\lambda x.x)((\lambda x.x + x)z))2 \end{aligned}$$

$$(\lambda x.x)((\lambda x.x + x)2) = (\lambda x.x + x)2 = 2 + 2 = 4$$

$$(\lambda z.((\lambda x.x + x)z))2 = (\lambda z.z + z)2 = 2 + 2 = 4$$

6.2 Состав отчета

- Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
- Текст LISP программы
- β -редукции

Варианты заданий

- 1 $((((\lambda xyz.x(yz))(\lambda x.x + x))(\lambda x.x \cdot x))3)$
- 2 $((((\lambda xyz.x(yz))(\lambda x.x \cdot x))(\lambda x.x + x))4)$
- 3 $((((\lambda xyz.x(yz))(\lambda x.x \cdot x))(\lambda x.x \cdot x))5)$
- 4 $((((\lambda xyz.x(yz))(\lambda x.x + 8))(\lambda x.9 \cdot x))6)$
- 5 $((((\lambda xyz.x(yz))(\lambda x.x))(\lambda x.x \cdot x))7)$
- 6 $((((\lambda xyz.x(yz))(\lambda x.x + x))(\lambda x.x + x))8)$
- 7 $(((((\lambda xyz.xzy))(\lambda xy.x \div y))3)9)$
- 8 $(((((\lambda xyz.xzy))(\lambda xy.x \div y))((\lambda x.x)3))9)$
- 9 $(((((\lambda xyz.xzy))(\lambda xy.x \div y))3)((\lambda x.x)9))$
- 10 $(((((\lambda xyz.xz(yz)))(\lambda xy.x + y))(\lambda x.x + 2)3)$
- 11 $(((((\lambda xyz.xz(yz)))(\lambda xy.x))(\lambda x.x)4))$
- 12 $(((((\lambda xyz.xz(yz)))(\lambda xy.y))(\lambda x.x + 2)5)$
- 13 $(((((\lambda xyz.xz(yz)))(\lambda xy.x \cdot y))(\lambda x.x + 2)6)$
- 14 $(((((\lambda xyz.xz(yz)))(\lambda xy.x + y))(\lambda x.x \cdot 2)7)$
- 15 $(((((\lambda xyz.xz(yz)))(\lambda xy.x - y))(\lambda x.x - 2)8)$

7 Лабораторная работа №5

Основные возможности Haskell

7.1 Цели

- Приобрести навыки работы с интерпретатором языка Haskell. Получить представление об основных типах языка Haskell. Научиться определять простейшие функции.

7.2 Задание

- 1 Изучить теоретические сведения
- 2 Выполнить задания в соответствии с вариантом

7.3 Теоретические сведения

7.3.1 Типы

- Типы `Integer` и `Int` используется для представления целых чисел, причем значения типа `Integer` не ограничены по длине.
- Типы `Float` и `Double` используется для представления вещественных чисел.
- Тип `Bool` содержит два значения: `True` и `False`, и предназначен для представления результата логических выражений.
- Тип `Char` используется для представления символов.

7.3.2 Списки

Чтобы задать список в Haskell, необходимо в квадратных скобках перечислить его элементы через запятую. Все эти элементы должны принадлежать одному и тому же типу. Тип списка с элементами, принадлежащими типу `a`, обозначается как `[a]`. Примеры: `[1,2]`; `['1','2','3']`.

Операции со списками Оператор `:` (двоеточие) используется для добавления элемента в начало списка. Его левым аргументом должен быть элемент, а правым - список:

```
> 1:[2,3]
[1,2,3] :: [Integer]
> '5':['1','2','3','4','5']
['5','1','2','3','4','5'] :: [Char]
> False:[]
[False] :: [Bool]
```

Listing 10: Пример

С помощью оператора `(:)` и пустого списка можно построить любой список:

```
> 1:(2:(3:[]))
[1,2,3] :: Integer
> 1:2:3:[]
[1,2,3] :: Integer
```

Listing 11: Пример

Элементами списка могут быть любые значения — числа, символы, кортежи, другие списки и т.д.

```
> [(1, 'a'), (2, 'b')]
[(1, 'a'), (2, 'b')] :: [(Integer, Char)]
> [[1, 2], [3, 4, 5]]
[[1, 2], [3, 4, 5]] :: [[Integer]]
```

Listing 12: Пример

Для работы со списками в языке Haskell существует большое количество функций. В данной лабораторной работе рассмотрим только некоторые из них.

- Функция `head` возвращает первый элемент списка.
- Функция `tail` возвращает список без первого элемента.
- Функция `length` возвращает длину списка.

Функции `head` и `tail` определены для непустых списков. При попытке применить их к пустому списку интерпретатор сообщает об ошибке. Примеры работы с указанными функциями:

```
> head [1, 2, 3]
1 :: Integer
> tail [1, 2, 3]
[2, 3] :: [Integer]
> tail [1]
[] :: Integer
> length [1, 2, 3]
3 :: Int
```

Listing 13: Пример

Для соединения (конкатенации) списков в Haskell определен оператор `++`.

```
> [1, 2] ++ [3, 4]
[1, 2, 3, 4] :: Integer
```

Listing 14: Пример

7.3.3 Функции

Рассмотрим пример:

```
square :: Integer -> Integer
square x = x * x
```

Listing 15: Пример

Первая строка (`square :: Integer -> Integer`) объявляет, что мы определяем функцию `square`, принимающую параметр типа `Integer` и возвращающую результат типа `Integer`. Вторая строка (`square x = x * x`) является непосредственным определением функции. Функция `square` принимает один аргумент и возвращает его квадрат.

В общем виде тип функции, принимающей n аргументов, принадлежащих типам t_1, t_2, \dots, t_n , и возвращающей результат типа a , записывается в виде $t_1 \rightarrow t_2 \rightarrow \dots \rightarrow t_n \rightarrow a$.

```
add :: Integer -> Integer -> Integer
add x y = x + y
```

Listing 16: Пример

Условное выражение В общем виде выглядит так:
if условие then выражение else выражение.

Функцию `signum`, вычисляющую знак переданного ей аргумента:

```
signum :: Integer -> Integer
signum x = if x > 0
            then 1
            else if x < 0
                  then -1
                  else 0
```

Listing 17: Пример

Следует обратить внимание на отступы. Именно по отступам компилятор определяет к какому `if` относится тот или иной `else`.

Условие в определении условного оператора представляет собой любое выражение типа `Bool`. Примером таких выражений могут служить сравнения. При сравнении можно использовать следующие операторы:

- `<`, `>`, `<=`, `>=` - эти операторы имеют такой же смысл, как и в языке Си (меньше, больше, меньше или равно, больше или равно);
- `==` - оператор проверки на равенство;
- `/=` - оператор проверки на неравенство.

7.4 Пример

Задание Написать функцию на языке Haskell возвращающую знак переданного целого числа.

```
signum :: Integer -> Integer
signum x = if x > 0
            then 1
            else if x < 0
                  then -1
                  else 0

main = print $ Prelude.signum 1
```

Listing 18: Пример

7.5 Состав отчета

- Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
- Текст программы на языке Haskell
- Результат работы программы на языке Haskell

7.6 Варианты заданий

- 1 Функция `max3`, по трем целым возвращающая наибольшее из них.
- 2 Функция `min3`, по трем целым возвращающая наименьшее из них.
- 3 Функция `sort2`, по двум целым возвращающая пару, в которой наименьшее из них стоит на первом месте, а наибольшее - на втором.

- 4 Функция `bothTrue :: Bool -> Bool -> Bool`, которая возвращает `True` тогда и только тогда, когда оба ее аргумента будут равны `True`. Не используйте при определении функции стандартные логические операции (`&&` , `||` и т.п.).
- 5 Функция `solve2::Double->Double->(Bool,Double)`, которая по двум числам, представляющим собой коэффициенты линейного уравнения $ax + b = 0$, возвращает пару, первый элемент которой равен `True`, если решение существует и `False` в противном случае; при этом второй элемент равен либо значению корня, либо `0.0`.
- 6 Функция `isParallel`, возвращающая `True`, если два отрезка, концы которых задаются в аргументах функции, параллельны (или лежат на одной прямой). Например, значение выражения `isParallel (1,1) (2,2) (2,0) (4,2)` должно быть равно `True`, поскольку отрезки $(1, 1) - (2, 2)$ и $(2, 0) - (4, 2)$ параллельны.
- 7 Функция `isIncluded`, аргументами которой служат параметры двух окружностей на плоскости (координаты центров и радиусы). Функция возвращает `True`, если вторая окружность целиком содержится внутри первой.
- 8 Функция `isRectangular`, принимающая в качестве параметров координаты трех точек на плоскости, и возвращающая `True`, если образуемый ими треугольник - прямоугольный.
- 9 Функция `isTriangle`, определяющая, можно ли их отрезков с заданными длинами x , y и z построить треугольник.
- 10 Функция `isSorted`, принимающая на вход три числа и возвращающая `True`, если они упорядочены по возрастанию или по убыванию.
- 11 Функция принимает два числа, и если их сумма чётна, то возвращает их разницу, иначе - сумму.
- 12 Функция принимает список и число, и добавляет число к списку.
- 13 Функция принимает список и число, и добавляет число к списку с конца.
- 14 Функция принимает 3 числа и возвращает список с этими числами
- 15 Функция принимает 2 числа, и возвращает наибольшее кратное двойке.

7.7 Лабораторная работа №6

Использование комбинаторов в языке Haskell

7.8 Цели

- Познакомиться с комбинаторами на языке Haskell

7.9 Задание

- 1 Построить комбинаторы i , b , k , c , w , s ;
- 2 функции $p\ x = x + 1$, $u\ x\ y = x + y$ (функции p и u могут быть использованы не более одного раза);
- 3 проверить их работоспособность;
- 4 построить выражение в соответствии с заданием.

7.10 Пример

Задание Построить выражение $v\ 6\ 7 = 7$

```
i x = x
k x y = x
s x y z = x z (y z)
b x y z = x (y z)
c x y z = x z y
w x y = x y y
```

```
p x = x + 1
u x y = x + y
```

```
main = print $ k i 6 7
```

Listing 19: Пример

7.11 Состав отчета

- Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
- Текст программы на языке Haskell
- Результат работы программы на языке Haskell

7.12 Варианты заданий

- 1 $v\ 6\ 7 = 7$
- 2 $v\ 6 = 8$
- 3 $v\ 5 = 10$
- 4 $v\ 4 = 9$
- 5 $v\ 3\ 5 = 9$
- 6 $v\ 4\ 7 = 5$
- 7 $v\ 3\ 4\ 5 = 9$

$$8 \ v \ 7 \ 2 \ 6 = 2$$

$$9 \ v \ 8 \ 2 = 3$$

$$10 \ v \ 8 \ 7 = 14$$

$$11 \ v \ 7 \ 9 = 14$$

$$12 \ v \ 2 = 4$$

$$13 \ v \ 9 = 18$$

$$14 \ v \ 5 = 11$$

$$15 \ v \ 2 \ 6 = 9$$

$$16 \ v \ 3 \ 5 = 4$$

$$17 \ v \ 8 \ 2 \ 5 = 7$$

$$18 \ v \ 7 \ 3 \ 6 = 3$$

$$19 \ v \ 8 \ 1 = 2$$

$$20 \ v \ 8 \ 6 = 12$$

$$21 \ v \ 4 \ 9 = 8$$

8 Лабораторная работа №7

Формализация предметной области для языка Пролог

8.1 Цели

- Познакомиться языком Пролог

8.2 Задание

- 1 Для соответствующей варианту предметной области составить 3-5 аксиом.
- 2 Составить несколько вопросов.
- 3 Записать на языке Пролог.

8.3 Пример

Задание

- Предметная область: раковина;
- описание предметной области: Раковина засоряется тем, что пропустит фильтр. Фильтр пропускает очистки морковки при использовании любых инструментов для чистки и очистки любого овоща очищенного картофелечисткой. Морковь почистили ножом, картофель - картофелечисткой;
- вопросы: Чем засорится раковина? Засорится ли раковина?.

Решение

Объявим предикаты

- $C(x, y)$ - чистить x инструментом y
- $F(x)$ - фильтр пропустит x
- $R(x)$ - раковина засорилась x

$$\begin{aligned} A1 &: F(x) \rightarrow R(x) \\ A2 &: C(x, \textit{kartochistka}) \rightarrow F(x) \\ A3 &: C(\textit{morkva}, y) \rightarrow F(\textit{morkva}) \\ A4 &: C(\textit{morkva}, \textit{nozh}) \\ A5 &: C(\textit{kartofan}, \textit{kartochistka}) \\ B1 &: R(x) \\ B2 &: R(\textit{kartofan}) \end{aligned}$$

Напишем программу на языке Пролог:

```
r(X) :- f(X).
f(X) :- c(X, 'kartochistka').
f('morkva') :- c('morkva', _).
c('morkva', 'nozh').
c('kartofan', 'kartochistka').
```

Listing 20: Пример

Зададим пару вопросов программе:

```
$ gprolog
GNU Prolog 1.2.18
By Daniel Diaz
Copyright (C) 1999–2004 Daniel Diaz
| ?- consult('rack.pl').
compiling /home/kdb/prolog/rack.pl for byte code...
/home/kdb/prolog/rack.pl compiled, 5 lines read - 739 bytes written, 25 ms

yes
| ?- r('morkva').

yes
| ?- r(X).

X = kartofan ? a

X = morkva

yes
| ?- trace.
The debugger will first creep — showing everything (trace)

yes

{trace}
| ?- r('kartofan').
    1      1      Call: r(kartofan) ?
    2      2      Call: f(kartofan) ?
    3      3      Call: c(kartofan,kartochistka) ?
    3      3      Exit: c(kartofan,kartochistka) ?
    2      2      Exit: f(kartofan) ?
    1      1      Exit: r(kartofan) ?

yes
{trace}
| ?- r(X).
    1      1      Call: r(_16) ?
    2      2      Call: f(_16) ?
    3      3      Call: c(_16,kartochistka) ?
    3      3      Exit: c(kartofan,kartochistka) ?
    2      2      Exit: f(kartofan) ?
    1      1      Exit: r(kartofan) ?

X = kartofan ? a
    1      1      Redo: r(kartofan) ?
    2      2      Redo: f(kartofan) ?
    3      3      Call: c(morkva,_69) ?
    3      3      Exit: c(morkva,nozh) ?
    2      2      Exit: f(morkva) ?
    1      1      Exit: r(morkva) ?

X = morkva

yes
{trace}
| ?-
```

Listing 21: Результат

8.4 Состав отчета

- Титульный лист (фамилия, группа, номер варианта, наименование работы, задание)
- Задание
- Аксиомы
- Вопросы
- Программа на языке Пролог

8.5 Варианты заданий

- 1 книжный шкаф
- 2 зоопарк
- 3 театр
- 4 цирк
- 5 телевизор
- 6 баня
- 7 таблица умножения
- 8 сдача экзамена
- 9 ноутбук
- 10 файловая система
- 11 правительства
- 12 система прав пользователей
- 13 продуктовый магазин
- 14 учебная аудитория
- 15 родственники
- 16 структура каталогов
- 17 арифметические действия
- 18 текст
- 19 гипертекст
- 20 холодильник
- 21 лес
- 22 огород