

Листок χ . Функциональное программирование. Часть 1

10 класс

Дата выдачи листка: 01.10.03.

«Того же, кто вас заморочил иной программой,
Я видел! Он давеча шел по проспекту с дамой,
Лицом лоснясь, как вяленый лещ.»

Михаил Щербаков

Идея: все можно выразить через функции. Циклы и переменные — блажь.

Напоминание 1 (неформальное). **Функцией** называется правило, которое каждому **аргументу** ставит в соответствие некоторое **значение**. Например, $sign(x)$ равно 1 для всех положительных чисел, -1 для отрицательных и 0 для 0. Особо стоит подчеркнуть, что у нас функция зависит только от своих аргументов, чего не скажешь, например, о функции $random(n)$ языка Pascal.

Напоминание 2. **Упорядоченной парой** (x, y) называется множество $\{x, \{x, y\}\}$. Таким образом, мы формально определяем, что один элемент выделен как первый. **Упорядоченной тройкой** (x, y, z) назовем пару $(x, (y, z))$. И так далее.

Задача 1. Придумайте другие способы формально выделить первый элемент в двухэлементном множестве. Придумайте другие определения тройки.

Задача 2. Дайте более формальное определение n -ки (по индукции).

Определение 1. **Кортеж** из n элементов — это n -ка. Следуя примеру Владимира Андреевича Успенского, будем придерживаться этого слова (о причине см. следующее определение).

Определение 2. Функцию от k элементов легко определить как функцию одного аргумента, который является кортежем из k элементов.

Определение 3. **Типы** — это множества, значения из которых принимают функции и их аргументы. Зафиксируем тип вещественных чисел **real**, целых — **integer**. Введем **логический** тип (**boolean**), состоящий из значений **истина** (обозначается 1) и **ложь** (обозначается 0). Будем считать, что функция может быть определена не для всех значений соответствующего типа. Например, тип аргумента и результата функции $\sqrt{x}$ (квадратный корень) — **real**, однако $\sqrt{-1.5}$ смысла не имеет.¹

Операции вполне естественно рассматривать как функции. Например, функция от двух аргументов $+$ возвращает их сумму. Отношения $<, >, =$ так же можно понимать как функции, которые по двум числам принимают значение «истина» или «ложь».

Функции принято писать слева от заключенных в скобки аргументов, но, следуя традиции, операции и отношения будем записывать в привычной форме, считая приоритеты стандартными. Например, « $+(2, \cdot(3, 4))$ » есть « $2 + 3 \cdot 4$ ».

Определение 4. Будем считать, что нам заданы все привычные числовые функции: $+, -, \cdot, /, \sin, \cos$ и т.д. Также в нашем распоряжении имеются функции от аргументов логического типа: $\&, \vee, \neg$...

Задача 3. Определите значения функций $\&, \vee, \neg$ на всех возможных значениях аргументов (то есть на всех возможных комбинациях 0 и 1).

Задача 4. Вычислите следующие выражения: a) $3/6 + 4/8$; b) $2 \cdot 2 = 4$.

Теперь мы будем писать функции, которые вычисляют что-нибудь содержательное. Например, функция для удвоения числа выглядит так: $f(x) = 2 \cdot x$. Мы разрешаем использование констант. Помните, что при определении функции можно использовать другие функции.

¹Терпение, мы и не такому, мы делить на 0 научимся!

Задача 5. Напишите

- a) функцию, которая уменьшает число в два раза;
- b) функцию от двух аргументов логического типа, которая принимает истинное значение только в случае, если аргументы равны (используйте только $\&$, $\vee$, $\neg$).

Определение 5. Определим функцию **if-then-else** от трех аргументов. Первый аргумент имеет логический тип. В случае, если он равен 1, функция возвращает второй аргумент, иначе — третий. Тип результата функции совпадает с типом второго и третьего аргументов.

Задача 6. Напишите функцию $\max(x, y)$, которая из двух чисел выбирает наибольшее.

Задача 7. То же для трех чисел.

Определение 6. Добавим возможность **рекурсии**, то есть определения функции через себя. Например, $g(n) = \text{if } (n > 0) \text{ then } n \cdot g(n - 1) \text{ else } 1$ — функция для вычисления факториала.

Задача 8. Реализуйте функции:

- a) для вычисления чисел Фибоначчи;
- b) для вычисления биномиальных коэффициентов (треугольник Паскаля);
- c) для нахождения НОДа;
- d) для нахождения НОКа;
- e) для тестирования числа на простоту (считаем заданной функцию вычисления остатка);
- f) для решения квадратного уравнения;
- g) еще для чего-нибудь полезного.

Hint: Помните, что часто сначала придется определить промежуточные функции.

Списки (лучше прочтите про типы — это интересней)

Определение 7. Теперь нам понадобятся рекурсивные структуры данных. **Список** — это конечная последовательность элементов одного типа (подумайте, как сказать это более строго). Например, $[1, 2, 3, 4, 5]$ или $[\text{«истина»}, \text{«ложь»}, \text{«ложь»}]$. Пусть нам заданы константа nil (пустой список) и функция $::$, которая по значению x и списку L получает список $x :: L$, в котором x добавлен в начало L . nil и $::$ называются **конструкторами** в том смысле, что с их помощью можно создать любой список. Например, $[1, 2, 3, 4, 5]$ есть краткая запись $1 :: 2 :: 3 :: 4 :: 5 :: nil$, а $[\text{«истина»}, \text{«ложь»}, \text{«ложь»}]$ на самом деле выглядит как $\text{«истина»} :: \text{«ложь»} :: \text{«ложь»} :: nil$. Будем обозначать тип списка $T \text{ list}$, где T — тип элементов. Например, *real list*.

Определение 8. Введем еще три функции: $isnil(L)$ принимает истинное значение на пустом списке и ложное в противном случае, $head(L)$ возвращает первый элемент непустого списка, а $tail(L)$ — его хвост, то есть все без первого элемента.

Задача 9. Вычислите $isnil(nil)$, $isnil(x :: L)$, $head(x :: L)$, $tail(x :: L)$, $head(nil)$ где x — некоторый элемент, а L — список элементов того же типа.

Теперь мы можем писать функции для работы со списками. Например, так можно реализовать поиск элемента: $search(x, L) = \text{if } isnil(L) \text{ then } 0 \text{ else if } head(L) = x \text{ then } 1 \text{ else } search(x, tail(L))$.

Задача 10. Реализуйте:

- a) слияние двух списков;
- b) функцию, меняющую порядок элементов списка на противоположный;
- c) разделение списка на два;
- d) слияние двух отсортированных списков в один отсортированный;
- e) сортировку слиянием;
- f) какой-нибудь другой алгоритм сортировки;
- g) еще что-нибудь полезное.

Задача 11. * Реализуйте то же самое на языке Pascal и сравните.

Листок χ . Функциональное программирование. Часть 2

10 класс

Дата выдачи листка: 01.09.03.

«Уймись, не греми кандалами, тебе повезет, повезет.
В твоей поднебесной программе никто ничего не поймет.»
Михаил Щербаков

Типы (лучше прочтите про списки — это полезней)

Задача 12. Какой тип имеет кортеж из двух вещественных чисел?

Определение 9. Для обозначения типа кортежа будем в скобках перечислять типы его элементов. Например, $(real, real)$.

Задача 13. Напишите типы аргументов реализованных вами функций.

Определение 10. **Типом функции** назовем тип ее аргумента и тип ее значения. Например, тип функции $\max$ записывается в форме $(real, real) \rightarrow real$.

Задача 14. Запишите типы реализованных вами функций.

Идея! Функции тоже могут быть аргументами. Кстати, это есть в языке Pascal. Функции могут быть результатом вычислений! Это есть в некоторых других языках. Например, можно рассматривать функцию $(\max(x)) (y)$ типа $real \rightarrow (real \rightarrow real)$. При таком порядке вычислений принято опускать скобки и писать $\max x y$. Другими словами, $\max 3 y$ есть функция от одного аргумента y , которая определяет максимум y и константы 3. Ура, теперь нам не нужны кортежи! Пока.

Задача 15. Напишите типы реализованных вами функций без использования кортежей.

Энергичные и ленивые вычисления (хотите вычислять лениво?)

Определение 11. Рассмотрим такую программу:

```
f(x,y) = if (x=0) then 0
          else y
g(x) = g(x)+1
f(0,g(1))
```

Чему будет равно выражение $f(0, g(1))$? При **энергичных вычислениях** сначала получают значения аргументов функции, затем вычисляют саму функцию. Таким образом наша программа зацикливается. Результат можно считать неопределенным. При **ленивых вычислениях** значения аргументов вычисляются только тогда, когда они действительно понадобятся. Тогда мы получаем, что $f(0, g(1))$ есть 0.

Задача 16. Какие вычисления имеют место в языке Pascal?

Ленивые вычисления можно использовать для работы с бесконечными данными. Например, легко получить список квадратов натуральных чисел:

```
getSquare(n)=(n*n)::getSquare(n+1)
getSquare(1)
```

Более сложный и содержательный пример — поиск простых чисел с помощью решета Эратосфена: Функция, порождающая список натуральных чисел:

```
getList(n)=n::getList(n+1)
```

Функция, которая возвращает 1, если первый аргумент есть делитель второго и 0 в противном случае:

```
isDivisor(x,y)=if (y mod x = 0) then 1
                else 0
```

Функция, которая удаляет из списка все элементы, кратные первому аргументу:

```
delete(x,L)=if isnil(L) then L
             else if isDivisor(x,head(L)) then delete(x,tail(L))
             else head(L)::delete(x,tail(L))
```

Основная функция – решето Эратосфена:

```
main(L)=if isnil(L) then L
        else head(L)::main(delete(head(L),tail(L)))
```

А теперь запускаем вычисление простых чисел:

```
main(getList(2))
```

Задача 17. Реализуйте:

- a) другой алгоритм поиска простых чисел;
- b) вычисление всех составных чисел;
- c) вычисление числа e , то есть функцию, порождающую список

$x_0 :: x_1 :: \dots :: x_n :: \dots$, где

$$x_n = \sum_{k=0}^n \frac{1}{k!};$$

- d) поиск всех совершенных чисел;
- e) еще что-нибудь интересное.