Информатика

Шилов Николай Вячеславович
Кандидат физико-математических наук, старший научный сотрудник Института систем информатики имени А.П. Ершова Сибирского отделения РАН.

Заметки о парадигмах программирования

Как известно, любую задачу можно решить несколькими способами. Но и саму задачу можно сформулировать совершенно по-разному и при этом получить один и тот же конечный результат. Мы рассмотрим так называемые «парадигмы программирования» на примере вычисления факториала натурального числа.

Что такое «парадигмы программирования»?

Слово «парадигма» имеет греческое происхождение и означает «пример», «образец». Говоря о парадигме программирования, мы имеем в виду разные способы постановки и решения одной и той же задачи. Именно в этом значении термин «парадигма программирования» был впервые употреблён Робертом Флойдом в 1978 году.

Заметим, что способ постановки задачи обычно определяет язык программирования, на котором будет реализовано её решение. Приведём классификацию языков программирования, появившихся в рамках той или иной парадигмы.

На начальном этапе становления программирования и информационно-вычислительных технологий (первые 10-15 лет начиная с 1950 г.) языков программирования было немного, и почти все они были языками *императивного* программирования. Наиболее известные представители этого семейства языков – это исторические FORTRAN, ALGOL-60 и классические PASCAL и C.

С конца 1960-х годов наряду с языками последовательного императивного программирования появились языки *параллельного* программирования, языки *декларативного* программирования (прежде всего функционального); в середине 1970-х – начале 1980-х годов родилось несколько новых «программирований», среди них – *логическое* и *объектно-ориентированное*.

Не останавливаясь подробно на всех типах, заметим, что широко из-

вестный исторический и классический логический язык только один – это Prolog. А вот семейство объектно-ориентированных языков знакомо многим по языкам C++ и Delphi.

В чём же смысл существования разных парадигм? Для того чтобы читатель мог понять разницу в постановке и, соответственно, в решении задач, мы предлагаем познакомиться с тремя из перечисленных пяти парадигм на примере вычисления факториала для натуральных чисел. Напомним, что факториалом натурального числа n называется произведение всех натуральных чисел от 1 до n:

$$n! = 1 \cdot 2 \cdot \ldots \cdot n.$$

Для написания программ мы будем использовать псевдокод, т. е. ориентированный на человека формат представления алгоритмов, а не код на каком-либо конкретном языке программирования.

«Императивный» факториал

«Императивный» – от латинского «командный». Императивная постановка задачи выглядит следующим образом: факториал натурального числа $N > 0$ – это произведение всех натуральных чисел от 1 до N. Императивный псевдокод состоит из команд (операторов над памятью), например, для вычисления факториала можно предложить следующий псевдокод низкого уровня[1]:

```
1: var n, f : integer;
2: input(n);
3: f:= 1;
4: if n = 1 then goto 8;
5: f:= f*n;
6: n:= n-1;
7: goto 4;
8: output(f);
9: end.
```

Команды преобразуют значения в «ячейках памяти» вычислительной машины (давайте считать, что любая ячейка способна хранить любое число в диапазоне от 0 до $N!$). Запуск императивной программы соответствует размещению её команд в памяти и последовательному исполнению команд, начиная с первой (команды с номером 1). В таблице 1 представлено вычисление факториала числа 3. Строки представляют собой последовательные моментальные состояния памяти некоторой «модельной» вычислительной машины. Эту таблицу надо понимать так. Строка, соответствующая первому моменту времени, описывает

результат исполнения команды с номером 1 (т. е. команды `1: var n, f : integer;`): она выделила две ячейки памяти под переменные n и f (в нашем случае это ячейки 3 и 4), остальные ячейки оставила свобод-

[1] Мы специально используем псевдокод уровня ассемблера, чтобы чётче проследить работу с памятью компьютера. Разумеется, этот же алгоритм можно было бы написать на классических императивных языках Pascal или C, или их объектно-ориентированных «наследниках» Delphi и C++. Но тогда работу с памятью проследить было бы сложнее.

ными и подготовила передачу управления команде со следующим по порядку номером (т. е. 2). Второму моменту времени соответствует строка, которая описывает результат выполнения команды с номером 2 (т. е. 2: `input(n);`): введённое значение (в нашем случае это 3) было занесено в ячейку, зарезервированную для n (т. е. ячейку 3), и подготовлена передача управления команде со следующим по порядку номером (т. е. 3).

Таблица 1

Номер ячейки → / Момент времени ↓	1 зарезервирована для номера исполненной команды	2 зарезервирована для номера следующей команды	3	4	...	...	...
1	1	2	выделена для n	выделена для f	Свободная память		
2	2	3	3	выделена для f	Свободная память		
3	3	4	3	1	Свободная память		
4	4	5	3	1	Свободная память		
5	5	6	3	3	Свободная память		
6	6	7	2	3	Свободная память		
7	7	4	2	3	Свободная память		
8	4	5	2	3	Свободная память		
9	5	6	2	6	Свободная память		
10	6	7	1	6	Свободная память		
11	7	4	1	6	Свободная память		
12	4	8	1	6	Свободная память		
13	8	9	1	6	Свободная память		
14	9	Свободная память					

В третий момент времени (см. соответствующую строку) в соответствии с командой с номером 3 (т. е. 3: `f:=1;`) произошло присвоение значения 1 ячейке, зарезервированной для f (т. е. ячейке 4), и подготовлена передача управления команде со следующим по порядку номером (т. е. 4).

В четвёртый момент времени была исполнена команда с номером 4 (т. е. 4: **`if`** `n = 1` **`then goto`** 8;), в результате чего значение, хранящееся в ячейке 3 (выделенной для n) было сравнено с 1, а так как значение было 3, то передачи управления команде с номером 8 не произошло, а была подготовлена переда-

ча управления команде со следующим по порядку номером (т. е. 4).

Подобным образом вычисления продолжаются до 13-го момента времени, когда после исполнения команды с номером 8 (т. е. 8: `output(f);`) произошёл вывод (на печать, экран и т. д.) значения 6, скопированного из ячейки, зарезервированной для f (т. е. ячейки 4), и подготовлена передача управления команде со следующим по порядку номером (т. е. 9). В четырнадцатый (последний) момент работы программы была исполнена команда с номером 9 (т. е. 9: **end**.), которая освободила всю память, включая ячейку 2, зарезервированную для номера следующей команды.

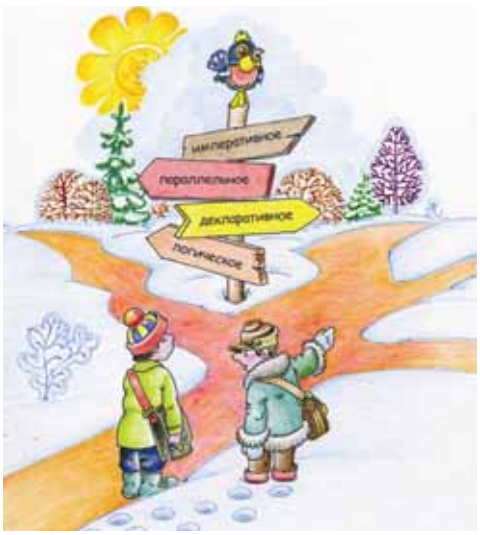
«Функциональный» факториал

В функциональном программировании постановка задачи и решение задачи – это определение функции, а вычисление – это цепочка равенств, где каждое следующее получается из предыдущего в результате однократного применения функции к аргументу (или аргументам). В частности, функциональная постановка задачи про факториал звучит следующим образом: факториал – это функция F, которая каждому натуральному числу сопоставляет натуральное число и удовлетворяет следующему тождеству: $F(x)=1$, если $x=1$, иначе $F(x)=$ $=x*F(x-1)$. Функциональный псевдокод состоит из одного определения функции F, но включает два случая:

$$F:1=1;$$

$$x=x*F(x-1).$$

Приведённая программа означает, что если конкретное значение аргумента функции F равно 1, то значение функции F от этого аргумента равно 1; в противном случае значение функции F от этого аргумента равно значению аргумента, умноженному на значение функции F от числа, которое на единицу меньше, чем аргумент. Никаких ячеек памяти для хранения значений в функциональном программировании нет, но есть память, в которой хранится сама функциональная программа. Вызов функциональной программы – это применение одной из функций программы к конкретному значению аргумента. Вычисление вызванной функциональной программы – это цепочка равенств. Эта цепочка начинается с вызова одной из функций этой программы с фактическим значением аргумента и заканчивается числом – значением этой функции на этом аргументе. Например, вычисление факториала числа 3 – это следующая цепочка равенств: $F(3)=$ (т. к. 3 не равно 1) $=3*F(3-1)=3*F(2)=$ (т. к. 2 не равно 1) $=3*\big(2*F(2-1)\big)=3*\big(2*F(1)\big)=$ (т. к. 1 равно 1) $=3*(2*1)=3*2=6$.



Функциональной парадигме лучше всего соответствуют функциональ-

ные языки программирования (среди них можно назвать популярный ныне Haskell). Но реализовать функциональное программирование можно и на императивных языках. Например, простейший способ переноса функциональных алгоритмов на язык Pascal – это представление каждого определения в виде определения функции, у которой все параметры – значения, а в теле не используются локальные переменные. Аналогично можно программировать и на языке С.

«Логический» факториал

В логическом программировании постановка задачи и её решение – это аксиоматическое определение отношения между аргументами и результатами, а вычисление – это поиск доказательства. Например, логическое определение факториала может состоять из двух аксиом, описывающих свойства отношения $P(m, n) =$ «m является факториалом n»:

- 1 является факториалом 1;
- m является факториалом n, если частное от деления нацело (m/n) является факториалом $(n-1)$.

Формально эти две аксиомы можно представить следующим образом:

$$P(1,1);\ \ P(m/n, n-1) \Rightarrow P(m, n);$$

первая из этих аксиом означает, что «число 1 является факториалом числа 1», а вторая – что «если частное от деления нацело m на n является факториалом числа $(n-1)$, то само число m является факториалом числа n».

Функциональный псевдокод записывается в виде:

$$P(1,1); \qquad (1)$$

$$P(m, n) :: P(m/n,\ n-1). \qquad (2)$$

Выражение типа (1) называется фактом и, как вы уже поняли, является утверждением в обычном смысле слова.

Выражение типа (2) называется правилом. В общем случае правило означает: если выражение слева выполняется (истинно), то выполняется и выражение справа. В левой и правой частях правила находятся логические выражения, зависящие от аргументов. Напомним, что логические выражения могут принимать только два значения: «истина» или «ложь». Выражения такого типа называются предикатами.

Как и в функциональном программировании, никаких ячеек памяти для хранения значений в логическом программировании нет, но есть память (или «база знаний»), в которой хранится сама логическая программа.

Вызов логической программы – это любой из предикатов этой программы, в который подставлены фактические значения некоторых аргументов. Значения остальных аргументов предстоит определить во время «вычисления» программы

В качестве примера вычислим факториал числа 3.

- Для доказательства $P(m, 3)$ надо доказать $P((m/3),\ 3-1)$, так как аргументы $P(m, 3)$ невозможно унифицировать с аргументами факта $P(1,1)$.
- Доказательство $P((m/3), 3-1)$ совпадает с доказательством $P((m/3), 2)$, так как $(3-1)$ есть 2.
- Для доказательства $P((m/3), 2)$ надо доказать $P(((m/3)/2), 2-1)$, так как аргументы $P((m/3), 2)$ невозможно унифицировать с аргументами

факта $P(1, 1)$.

- Доказательство $P\big(((m/3)/2), 2-1\big)$ совпадает с доказательством $P\big(((m/3)/2), 1\big)$, так как $(2-1)$ есть 1.
- Так как аргументы $P\big(((m/3)/2), 1\big)$ возможно унифицировать с аргументами факта $P(1, 1)$ посредством подстановки 6 в качестве значения переменной m, то таким образом доказано $P(6, 3)$, т. е. что «число 6 является факториалом числа 3», чем завершается вычисление примера.

Возникает естественный вопрос: на каком языке лучше всего писать логиче-ские программы? Ответ краток: на Prolog. А можно ли логически программировать на императивных или функциональных языках? Можно, но уже не так просто: сначала придётся реализовать процедуру поиска доказательства, а потом – логически программировать.

Вместо заключения

Итак, на примере вычисления факториала были показаны особенности трёх парадигм программирования: императивной, функциональной и логической. Надеюсь, вы почувствовали разницу. Теперь самое время дать задание для самостоятельной работы.

Задача. Числа Фибоначчи – это последовательность целых чисел $a_0, a_1, \ldots, a_{n-1}, a_n, a_{n+1}$ такая, что $a_0 = a_1 = 1$ и $a_{n+1} = (a_{n-1} + a_n)$ для любого $n > 0$; таким образом, первые члены этой последовательности – это 1, 1, 2, 3, 5, 8, 13, 21, 34, ... Опишите императивный, функциональный и логический варианты постановки и решения задачи вычисления по входному значению $n \geq 0$ значения n-ого числа Фибоначчи a_n. Присылайте свои вопросы, императивную, функциональную и логическую формулировки и псевдокод решений по электронной почте на адрес shilov@iis.nsk.su.

Работа подготовлена в рамках проекта «Исследование и классификация парадигм компьютерных языков» (РФФИ 08-01-00899а).

Литература

1. *Душкин Р.В.* Функциональный подход в программировании// Потенциал.– 2009. – №8. – С. 47–55.

2. *Душкин Р.В.* Задача о ранце// Потенциал.– 2009. – №9. – С. 48–55.

3. *Кун Т.* Структура научных революций. Издательство АСТ, 2003.

4. *Флойд Р.* О парадигмах программирования. В кн.: Лекции лауреатов премии Тьюринга. – М: Мир, 1993.

5. *McCarthy J.* Recursive Functions of Symbolic Expressions and Their Computation by Machine. Communications of ACM. 1960, v.3 (4).