Может возникнуть вопрос, как вызывающая программа видит подпрограмму, если подпрограмма описана вне вызывающей программы. Ответ очень прост: описание подпрограмм является частью кода программы, именуемом разделом подпрограмм. А сам раздел подпрограмм помещается текстуально выше кода основной программы.

5.9. Лямбда-выражения

Вот мы и подошли к материалу, который пока что в школьной информатике, как правило, не рассматривают. Мы тоже на будем подробно и научно рассматривать эти «лямбды», а обратимся к практике их использования.

Лямбда-выражения (или просто лямбды) – термин ***функционального программирования***. Они с успехом применяются вместо уже привычных вам процедур и функций. Собственно, они и есть процедуры или функции, только безымянные. Вернитесь к определению подпрограммы на первой странице главы 5. Та самая фраза «или идентифицированный иным образом фрагмент программного кода» как раз и подразумевает лямбды.

> Лямбда-выражение представляет собой некоторое безымянное выражение, отражающее функциональную зависимость. Упрощенно, на его основе компилятор строит функцию, некоторым образом идентифицирует ее и подменяет этим идентификатором лямбда-выражение.

В коде программы лямбда-выражения легко найти по характерной паре символов ->. Это операция, которую называют по-разному, но мне нравится фраза «переходит в …». Вот пример:

```
x -> 3 * x * x - 6.3 * x + Sqrt(x)
```

Данное лямбда-выражение читается как «x переходит в $3x^2+6.3x+\sqrt{x}$». По сути, это запись функции с параметром x, которая возвращает значение выражения $3x^2+6.3x+\sqrt{x}$. Помните, в комедии «Бриллиантовая рука»: – Брюки превращаются … в элегантные шорты. Если бы там знали о лямбдах, могли сказать «переходят в …» вместо «превращаются».

```
(x, y) -> if x > y then x else y;
```

А это лямбда-функция с двумя параметрами и возвращает она максимальный из них.

```
x -> begin Write('Значение равно ', x:0:5) end;
```

Эта лямбда не возвращает значения. Возможно вы уже поняли, что так записываются лямбда-процедуры.

Чем интересы лямбды? Во-первых, их не надо заранее описывать вне вызывающей программы как отдельные функции и процедуры, а можно записывать прямо в коде программы по мере надобности. Во-вторых, как будет показано в следующих главах, лямбды очень сильно упрощают программирование многих задач, являясь параметрами (да-да, именно параметрами) подпрограмм. А еще, лямбду можно присвоить переменной и с этого момента обращаться к ней, как к именованной.

Если вы внимательно знакомились с материалом, должен был возникнуть вопрос: а как же типы? В лямбде x -> x * x какой тип имеет x? Целый, вещественный, логический? В PascalABC.NET для описания типов в лямбда выражениях существует свой синтаксис.

```
begin
  var x: integer-> integer := t -> t * t;
  Print(x(7)); // 49
  x := t -> t * t * t;
  Print(x(7)); // 343
  x := t -> 3 * t - 1;
  Print(x(7)) // 20
end.
```

Посмотрите на описание. Какой тип имеет переменная *x*? Может быть, вам это покажется странным, но ее тип **integer -> integer** ! Т.е. это лямбда-функция с параметром типа **integer**, возвращающая результат типа **integer**. В остальной части программы функция x переопределяется, поэтому один и тот же вызов x(7) порождает разные результаты.

Можно ли было в программе написать x := t -> t**3 ? Нет! Операция ** возвращает тип **real**, а в соответствии с описанием должен быть тип **integer**. Поэтому при компиляции будет выдана ошибка «Нельзя преобразовать тип real к integer».

Но если сделать описание

```
var x: integer -> real := t -> t * t;
```

в правой части лямбда-выражений можно указывать любые арифметические выражения, приводимые к **real**.

Задача 5.7
Написать процедуру, выводящую таблицу значений произвольной функции одного аргумента на интервале [a;b] с шагом h.

```
procedure Tab(a, b, h: real; f: real -> real);
begin
  var x := a;
  while x <= b + h / 2 do
  begin
    Writeln(x:10:3, f(x):20:12);
    x += h
  end
end;

begin
  Tab(-5, 3, 1, x -> x * x);
  Writeln;
  Tab(-0.9 * Pi, Pi, Pi / 8, x -> Sqr(Sin(x)) + Sqrt(Abs(x)))
end.
```

В процедуре *Tab* последним параметром определена лямбда-функция. В основной программе указаны два вызова *Tab*, где конкретизируются табулируемые функции. Вот такие они – лямбды!