12 Fundamentals of functional programming

12.1 Functional programming paradigm

Learning objectives:

- *Function as process*
- *Function as object*
- *Function, f, has a function type, $f: A \to B$ where the type is $A \to B$.*
- *A is the argument type, and B is the result type.*
- *The set A is called the domain and the set B is called the co-domain.*
- *The domain and co-domain are always subsets of objects in some data type.*

Key principle

Function as process:
A function is a rule that tells us how to transform some information into some other information.

Function as object:
The function is a thing in its own right.

12.1.1 Function type

What is a function?

Loosely speaking, a function is a rule that, for each element in some set *A* of inputs, assigns an output chosen from set *B* but without necessarily using every member of *B*.

For example, the function *f*

$$f: \{0,1,2,3\} \to \{0,1,2,3,4,5,6,7,8,9\}$$

maps 0 to 0, 1 to 1, 2 to 4 and 3 to 9 when the rule is: output the square of the input.

Function as process

In **function as process**, a function is a rule that tells us how to transform some information into some other information, e.g. the integer 2 into its square 4.

Function as object

In **function as object**, the function is a thing in its own right.

For example, a pencil sharpener is an object. If the focus of attention is a pencil then the pencil sharpener just represents a process - sharpening pencils, input: unsharpened pencil; output: sharpened pencil.

In the **function as process** view, we are applying the function *sharpen* to pencils; it's the pencil that counts. But we can also think about the pencil sharpener as a thing in its own right, when we empty it of pencil shavings, or worry about whether its blade is sharp enough. This is the **function as object** view.

Questions

A function *f*

$$f: \{0,1,2,3\} \to \{0,1,2,3, ..., 25, 26, 27\}$$

maps 0 to 0, 1 to 1, 2 to 8, 3 to 27.

1 What is the rule?

A function *f*

$$f: \{0,1,2,3\} \to \{0,1,2,3, 4, 5, 6\}$$

maps 0 to 0, 1 to 2, 2 to 4, 3 to 6.

2 What is the rule?

Questions

3 For each of the following what is the function as process and what is the function as object?

(a) A single sheet of A4 paper containing text is placed in the machine whose action is to produce a printed copy of the sheet.

(b) A kitchen tool is used to remove skin from potatoes.

Key principle

Function type:
A function f which takes an argument of type A and returns a result of type B has a function type which is written
$A \rightarrow B$

What is a function type?

Just as data values (e.g. 6, 9.1, True) have types (integer, real, Boolean respectively) so do functions. Function types are important because they state what type of argument a function requires and what type of result it will return.

A function f which takes an argument of type A and returns a result of type B has a function type which is written

$$A \rightarrow B$$

To state that f has this type, we write

$$f : A \rightarrow B$$

For example,

1) $squareroot : real \rightarrow real$

2) $square : integer \rightarrow integer$

The function named *squareroot* applied to an argument of data type *real* produces a result of data type *real*, e.g.

$$squareroot\ (4.0) \rightarrow 2.0$$

The function named *square* applied to an argument of data type *integer* produces a result of data type *integer*, e.g.

$$square\ (2) \rightarrow 4$$

Domain and co-domain

If $f : A \rightarrow B$ is a function from A to B we call the set A, the domain of f, and the set B the co-domain of f. The domain and co-domain are always subsets of objects in some data type. For example, if A is a subset of domain data type *integer* then its values might be 0, 1, 2, 3, ..., 149, 150. Often it is just convenient to use the data type directly,

$$square : integer \rightarrow integer$$

The function *square* then has an argument type, *integer* and a result type, *integer* even though in practice a subset of integers only will be used.

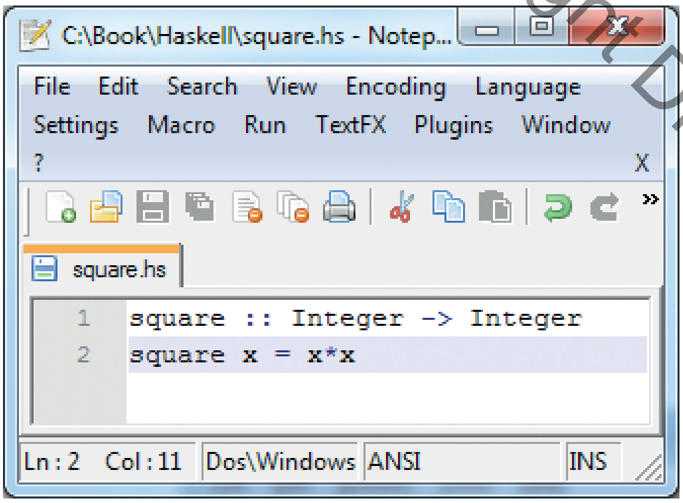
Key concept

Domain and co-domain:
If $f : A \rightarrow B$ is a function from A to B, we call the set A, the domain of f, and the set B the co-domain of f.

Practical Activity

Use a text editor such as NotePad++ to write Haskell programs. Save these Haskell programs using extension .hs.

Figure 12.1.1.1 shows NotePad++ being used to create a function named *square* with one parameter *x* of data type *Integer* and a body *x*x*. This file has been saved with filename `square.hs` in folder `c:\book\haskell`.



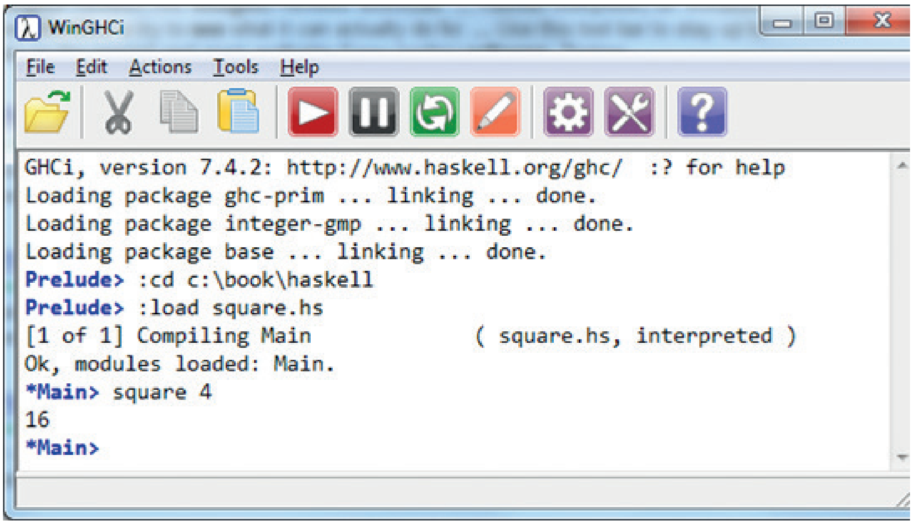
```
square :: Integer -> Integer
square x = x*x
```

Figure 12.1.1.1 NotePad++ editor showing square.hs

The :: operator (read as *has type*) is used in Haskell to express what type an expression has.

Integer is the type of mathematical integers (*int* could have been used and is the type of integers that fit into a word on the computer - this will vary from computer to computer).

Launch WinGHci if you are using a machine running the Windows operating system (ghci on Linux-based machines). The WinGHci window is shown in *Figure 12.1.1.2*.



```
GHCi, version 7.4.2: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude> :cd c:\book\haskell
Prelude> :load square.hs
[1 of 1] Compiling Main             ( square.hs, interpreted )
Ok, modules loaded: Main.
*Main> square 4
16
*Main>
```

Figure 12.1.1.2 WinGHCi showing square.hs loaded, compiled and run

At the Prelude prompt (Prelude>) type the command to change to a specified folder.

`:cd c:\book\haskell` followed by <return>.

Commands begin with a colon, i.e. :

Now load the file containing the program defining the function *square*.

At the Prelude prompt type

`:load square.hs` followed by <return>.

WinGHCi will perform a compilation of a module called `Main` in order to run `square.hs` interactively.

If there are no errors loading and compiling the Prelude prompt will be replaced by the prompt `*Main`.

At the `*Main` prompt, type

`square 4` followed by <return>.

The correct answer, 16, is displayed.

To return to the Prelude prompt, type :module or :m

In this chapter you have covered:

- Function as process
- Function as object
- Function, *f*, has a function type, *f*: *A* → *B* where the type is *A* → *B*.
- *A* is the argument type, and *B* is the result type.
- *A* is called the domain and *B* is called the co-domain.
- The domain and co-domain are always subsets of objects in some data type.