2 BASIC TYPES

In the last chapter we looked at sets and some of their properties. We noted that elements in a set are similar. It is this similarity that we focus on in this chapter.

To help describe the world about us, we classify objects into sets called types. We go on to look at how we can represent any element in a set, no matter how large that set is. We look at the properties of variables.

We see the importance of types in detecting errors and inconsistencies.

2.1 BASIC TYPES

There is just one inbuilt type that is part of the Z Notation: it is $\mathbb{Z}$, the set of all integers. We define all other types ourselves. We are in charge here.

We use a *basic type* when we want to just focus on essentials and hold off looking at details. For example, a person has a name, a date of birth and an address. If we do not need to be concerned with these details we introduce the type *PERSON*, the set of all possible people. If we do not need to bother with days, months, years and calendars (e.g. Chinese, Bengali and Gregorian) we introduce the type *DATE*, the set of all possible dates. If we do not need to bother with house number, street, city and postcode, we introduce the type *ADDRESS*, the set of all postal addresses. Focussing on essentials is called *abstraction*.

To define basic types of our own choosing we just list them (in alphabetical order) as given sets between square brackets, and explain what we mean by them. For example:

We introduce as given sets *ADDRESS*, *DATE* and *PERSON*. *ADDRESS* is the set of all postal addresses anywhere. *DATE*, the set of all possible dates in all possible calendars. *PERSON* is the set of all possible people.

$$[\ ADDRESS, DATE, PERSON\ ]$$

EXERCISE 2.1

1 Explain what is meant by the term abstraction. Illustrate your explanation with an example.

2 We are looking at defining a video rental system. A video has a title and a subject by which it is classified. We do not need to know anything about the internal details of titles and subjects. Introduce titles and subjects as given sets.

3 Each room in a hotel is given a number or a name. No two rooms have the same number or name. Since the number or name uniquely identifies a room, introduce room id as a given set.

4 A college offers many courses. Each course comprises one or more subjects. Introduce courses and subjects as given sets.

5 Each motor vehicle is uniquely identified by its registration mark, and is registered by just one owner at one address. Introduce owner, address and registration mark as given sets.

2.2 DECLARATIONS

PERSON, the set of all possible people on this planet, is pretty large. If we want to refer to just one of them we write

aPerson : *PERSON*

aPerson represents just one, any one, of all the possible elements in *PERSON*.

aPerson : *PERSON* is an example of a *declaration.* A declaration has two parts.

To the right of the colon is the name of a set; this name is *PERSON*.

To the left of the colon is a name for any element from that set; this name is *aPerson*. Since we do not necessarily know which element *aPerson* represents, *aPerson* is called a *variable*.

A variable has a *name* (e.g. *aPerson*) a *type* (e.g. *PERSON*) and a *value* taken from the type.

Look at this declaration.

aPersonsAge : 0..125

The name of the variable is *aPersonsAge*. Its type is $\mathbb{Z}$ because the values 0, 1, 2, and so on up to 125 are all elements from the larger set $\mathbb{Z}$. We think of a type as an inclusive set. Each element in the same set has the same type.

We cannot make a variable declaration if its type has not yet been introduced. For example, we cannot declare

$$aRoom : ROOM$$

before introducing *ROOM* as a given set by writing [*ROOM*] and describing what we mean by *ROOM*.

We define it before we use it. This is known as the *definition before use* principle. But note that the type $\mathbb{Z}$ is pre-defined; it is part of the Z notation. So you do not define $\mathbb{Z}$ before you use it.

EXERCISE 2.2

1 An employee has an annual salary that is always a whole number of pounds. Declare a variable to represent an employee's annual salary.

2 My car's thermometer displays air temperature to the nearest 0.5 degree Celsius. Thinking that 18.5 degrees C could be represented by 185, and 19 degrees by 190, declare a variable to represent the temperature displayed by my car's thermometer.

3 The printable characters found on a keyboard include letters of the alphabet, e.g. a, b, c, X, Y, Z, digits e.g. 1, 2, 3, symbols e.g. /, >, &, and white space such as tab and spacebar. Introduce a suitable type and declare a variable that could represent one of these printable characters.

4 In an office supplies catalogue, items of stationery are identified by their catalogue number. Introduce suitable types and declare variables that could represent items of stationery and their catalogue numbers.

5 Snakes and Ladders is a game played on a board of 100 squares, numbered 1 to 100. Declare a variable that could represent the position of a square on the board.

2.3 NAMING VARIABLES

We choose our own names for variables. We choose *descriptive* names whenever clarity is important. For example.

aPersonsAge : 0..125 rather than *a* : 0..125

We choose a single letter when there is no doubt about clarity (but readers often appreciate more descriptive names).

p : *PERSON* is ok but *aPerson* : *PERSON* might be preferable.

We follow the convention that names we choose for our variables:

- start with a lower case letter - *person* not *Person*
- are written entirely in lower case except the first letter of each word in the name - *aPersonsAge* not *apersonsage*
- do not contain spaces, underscores or hyphens - *anAddress* not *an Address*

Writing variable names in a mixture of lower and upper case (capital) letters helps us to tell them apart from type names, which are written entirely in upper case.

EXERCISE 2.3

Evaluate the variable names shown below against the convention for naming variables described in section 2.3 above.

1 *numberOfPersons* : 0..5

2 *NumberOfPersons* : 0..5

3 *number Of Persons* : 0..5

4 *NUMBEROFPERSONS* : 0..5

5 *n* : 0..5

2.4 CONSISTENCY

Every Z object is a type of one kind or another. This is important because it helps us discover inconsistencies in what we write.

For example, given the type

$$[\ PERSON\ ]$$

and the declarations

$$p, q : PERSON$$
$$r : \mathbb{Z}$$

$p = q$ is *consistent* because both p and q are variables of type *PERSON*.

But to say $p = r$ is a nonsense because p is of type *PERSON* and r is of type integer. How can you say that a person and an integer are the same kind of object?

EXERCISE 2.4

1 Comment on the validity of the expression $\{ red, blue, green \} = \{ 1, 2, 3 \}$

2 Given *CHARACTER* is the set of all printable characters found on any computer keyboard for any country,

$$[\ CHARACTER\ ]$$

and the declaration

$$ch : CHARACTER$$

comment on the validity of the expression

$$ch \in \mathbb{Z}$$

3 Given the declaration

$$capacity : \mathbb{Z}$$

comment on the validity of the expression

$$capacity = 5$$

REVIEW

We have seen that a basic type is a given set name. We think of a type as being an inclusive set. We noted that, in Z, all the elements of a set belong to the same type. We looked at declarations. A declaration introduces a variable and associates it with a type. We looked at the importance of types in reducing inconsistencies and errors. We shall look at composite types such as set types and Cartesian product types in later chapters. Next, we look at the integer type.

BIBLIOGRAPHY

BOTTACI L & JONES J. 1995 *Formal Specification Using Z* Thompson pp 139
JACKY J. 1997 *The Way of Z* Cambridge University Press pp 64
NORCLIFFE A. & SLATER G. 1991 *Mathematics of Software Construction* Ellis Horwood pp 43
SPIVEY J.M. 1992 *The Z Notation* Prentice Hall pp 7, 24, 51
WOODCOCK J. & DAVIES J. 1996 *Using Z: Specification, Refinement and Proof* Prentice Hall pp 70