Records and user-defined types

Ref As Pseudocode

Defining a record type

An array holds many values of **one** type. Often you need to keep **different** types together: a student's name (`STRING`), mark (`INTEGER`) and form group (`CHAR`). A **record** does this. Each item inside a record is a **field**.

A record is a **user-defined data type**: you design it first, then declare variables of that type, just as you would with `INTEGER`.

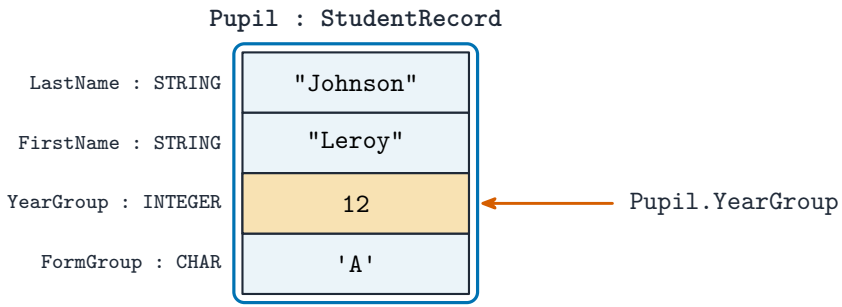
```
TYPE <identifier>
   DECLARE <field1> : <data type>
   DECLARE <field2> : <data type>
   ...
ENDTYPE
```

```
TYPE StudentRecord
   DECLARE LastName : STRING
   DECLARE FirstName : STRING
   DECLARE YearGroup : INTEGER
   DECLARE FormGroup : CHAR
ENDTYPE

DECLARE Pupil : StudentRecord
Pupil.LastName ← "Johnson"
Pupil.FirstName ← "Leroy"
Pupil.YearGroup ← 12
Pupil.FormGroup ← 'A'
OUTPUT Pupil.FirstName, " ", Pupil.LastName, " is in ", Pupil.YearGroup,
↪  Pupil.FormGroup
```

- `TYPE … ENDTYPE` only **describes** the record. It does not make a variable.
- `DECLARE Pupil : StudentRecord` makes a variable of the new type.
- Reach one field with **dot notation**: `<variable>.<field>`, as in `Pupil.YearGroup`.
- A field can be used anywhere a normal variable can: in `INPUT`, `OUTPUT`, conditions and assignments.

exercises.lol exercises.com.cn



A record groups fields of different types; dot notation reaches each one

Copying a whole record

Two variables of the same record type can be assigned in one line. Every field is copied.

```
TYPE Point
   DECLARE X : INTEGER
   DECLARE Y : INTEGER
ENDTYPE

DECLARE Start : Point
DECLARE Copy : Point
Start.X ← 3
Start.Y ← 7
Copy ← Start
Copy.X ← 10
OUTPUT "Start is (", Start.X, ", ", Start.Y, ")"
OUTPUT "Copy is (", Copy.X, ", ", Copy.Y, ")"
```

Changing `Copy.X` afterwards does not change `Start`: they are two separate records.

Common mistakes

- Using the **type** name instead of the variable: `StudentRecord.LastName ← "Lee"`. Write `Pupil.LastName`.
- `OUTPUT Pupil` to show a whole record. Output each field you need.
- Forgetting `DECLARE` in front of each field inside `TYPE … ENDTYPE`.

Arrays of records

An array whose elements are records is a very common data structure: it is like a table where each row is one record and each column is one field.

```
TYPE Book
   DECLARE Title : STRING
   DECLARE Pages : INTEGER
   DECLARE OnLoan : BOOLEAN
ENDTYPE

DECLARE Library : ARRAY[1:3] OF Book
DECLARE i : INTEGER
Library[1].Title ← "Dune"
Library[1].Pages ← 412
Library[1].OnLoan ← FALSE
Library[2].Title ← "Emma"
Library[2].Pages ← 474
Library[2].OnLoan ← TRUE
Library[3].Title ← "Holes"
Library[3].Pages ← 233
Library[3].OnLoan ← FALSE

FOR i ← 1 TO 3
   IF Library[i].OnLoan = FALSE THEN
      OUTPUT Library[i].Title, " (", Library[i].Pages, " pages) is
      ↪ available"
   ENDIF
NEXT i
```

- The index comes **first**, then the field: `Library[i].Title`.
- Read it from left to right: "in `Library`, element `i`, field `Title`".

Records instead of parallel arrays

Without records, you would need three separate arrays (`Titles`, `Pages`, `OnLoan`) that share one index. These are called **parallel arrays** . They work, but it is easy to sort one array and forget the others. An array of records keeps each book's data together.

Common mistakes

- `Library.Title[i]`. The index belongs to the array: `Library[i].Title`.
- Declaring the array before the `TYPE` that it uses. Define the type first.

exercises.lol exercises.com.cn

Other user-defined types

The 9618 guide describes more user-defined types. They are **A-Level (Paper 3)** content, but you will see them in the same guide, so it helps to recognise them. They do not run on this page.

Enumerated types

An **enumerated type** lists every value a variable of that type may have.

```
TYPE Season = (Spring, Summer, Autumn, Winter)
DECLARE ThisSeason : Season
ThisSeason ← Spring
```

The values are names, **not** strings: there are no quotes around `Spring`.

Pointers and sets

- A **pointer** type holds a memory address. `TYPE TIntPointer = ^INTEGER` declares it; `^` before a variable gives its address, and `^` after a pointer gives the value it points to.
- A **set** type holds a group of values with no order: `TYPE LetterSet = SET OF CHAR`.

For AS, focus on **records** and **arrays of records**: they appear in Paper 2 questions again and again.

Practice

1. Define a record type `Car` with the fields `Registration` (string), `Year` (integer) and `Price` (real). Declare an array of 3 cars, fill it, and output the registration of every car that costs less than 5000.

<details> <summary>Model answer</summary>

```
TYPE Car
   DECLARE Registration : STRING
   DECLARE Year : INTEGER
   DECLARE Price : REAL
ENDTYPE

DECLARE Cars : ARRAY[1:3] OF Car
DECLARE i : INTEGER
Cars[1].Registration ← "AB12 CDE"
Cars[1].Year ← 2015
Cars[1].Price ← 4500.0
Cars[2].Registration ← "FG34 HIJ"
Cars[2].Year ← 2019
Cars[2].Price ← 9800.0
Cars[3].Registration ← "KL56 MNO"
Cars[3].Year ← 2012
Cars[3].Price ← 2950.0
FOR i ← 1 TO 3
   IF Cars[i].Price < 5000.0 THEN
      OUTPUT Cars[i].Registration
   ENDIF
NEXT i
```

It prints `AB12 CDE` and `KL56 MNO`.
</details>

exercises.lol exercises.com.cn