记录与用户自定义类型

Ref As Pseudocode

定义记录类型

数组存放许多个**同一种**类型的值。但我们常常需要把**不同**类型的数据放在一起：一个学生的姓名（STRING）、分数（INTEGER）和班级（CHAR）。**记录**（record）就能做到这一点。记录里的每一项叫作一个**字段**（field）。

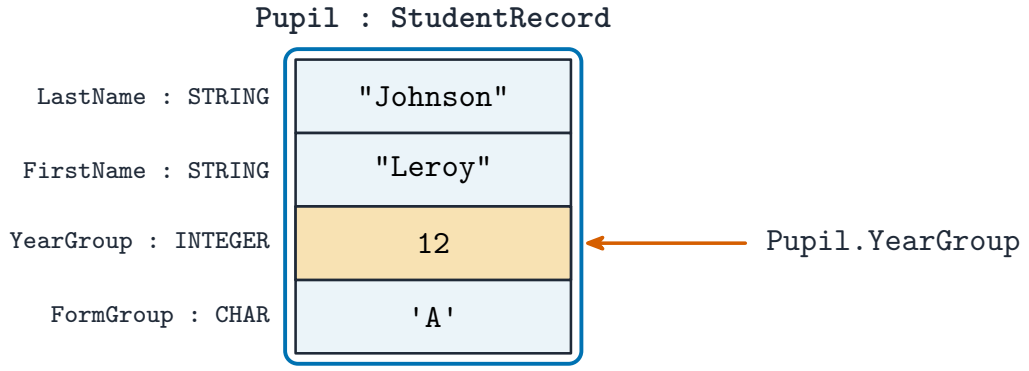
记录是一种**用户自定义数据类型**（user-defined data type）：你先设计好它，然后像使用 INTEGER 一样，声明这种类型的变量。

```
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 只是**描述**记录的样子，并不会创建变量。
- DECLARE Pupil : StudentRecord 才创建一个这种新类型的变量。
- 用**点号表示法**（dot notation）访问某个字段：<variable>.<field>，例如 Pupil.YearGroup。
- 字段可以用在普通变量能用的任何地方：INPUT、OUTPUT、条件和赋值。



记录把不同类型的字段组合在一起；用点号访问每个字段

■ 复制整条记录

同一记录类型的两个变量可以用一行来赋值，每个字段都会被复制。

```
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, ")"
```

之后修改 `Copy.X` 不会改变 `Start`：它们是两条独立的记录。

常见错误

- 用**类型**名代替变量名：`StudentRecord.LastName ← "Lee"`。应写 `Pupil.LastName`。
- 用 `OUTPUT Pupil` 显示整条记录。要分别输出你需要的每个字段。
- 在 `TYPE … ENDTYPE` 里漏写每个字段前面的 `DECLARE`。

记录数组

元素是记录的数组是一种非常常见的数据结构：它就像一张表格，每一行是一条记录，每一列是一个字段。

```
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
```

- 先写索引，再写字段：`Library[i].Title`。
- 从左往右读：“ 在 `Library` 中，第 `i` 个元素的 `Title` 字段 ”。

■ 用记录代替并行数组

如果没有记录，就需要三个共用同一个索引的独立数组（`Titles`、`Pages`、`OnLoan`），这叫作**并行数组**（parallel arrays）。它们也能用，但很容易只给一个数组排了序，却忘了其他数组。记录数组则让每本书的数据始终在一起。

常见错误

- 写成 `Library.Title[i]`。索引属于数组：应写 `Library[i].Title`。
- 在定义 `TYPE` 之前就声明使用它的数组。要先定义类型。

其他用户自定义类型

9618 指南还介绍了其他几种用户自定义类型。它们属于 **A-Level（Paper 3）** 的内容，但出现在同一份指南里，认识它们会有帮助。这些例子在本页上不能运行。

■ 枚举类型

枚举类型（enumerated type）列出这种类型的变量可以取的全部值。

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

这些值是名字，**不是**字符串：`Spring` 两边没有引号。

■ 指针与集合

- **指针**（pointer）类型存放一个内存地址。`TYPE TIntPointer = ^INTEGER` 声明这种类型；`^` 写在变量前面表示取它的地址，写在指针后面表示取它所指向的值。
- **集合**（set）类型存放一组没有顺序的值：`TYPE LetterSet = SET OF CHAR`。

AS 阶段请把重点放在**记录**和**记录数组**上：它们在 Paper 2 的题目中反复出现。

■ 练习

1. 定义一个记录类型 `Car`，字段为 `Registration`（字符串）、`Year`（整数）和 `Price`（实数）。声明一个含 3 辆车的数组并填入数据，然后输出所有价格低于 5000 的车的车牌号。

<details> <summary> 参考答案 </summary>

exercises.lol exercises.com.cn

```
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
```

输出 AB12 CDE 和 KL56 MNO。

</details>