

Solutions

Each answer shows the steps, then the **result**.

2.1

- for the date, a **record** (a user-defined composite type)
- because a date has parts of its own – a day, a month and a year – which belong together in one field
- for the team, an **enumerated** type, because the value is exactly one of a fixed, known list of names, and declaring it that way lets the compiler reject anything else

2.2

```
TYPE DateType
  DECLARE Day : INTEGER
  DECLARE Month : INTEGER
  DECLARE Year : INTEGER
ENDTYPE
```

- TYPE DateType ... ENDTYPE
- three DECLARE lines inside it
- all three of type INTEGER

2.3

```
TYPE MemberType
  DECLARE Name : STRING
  DECLARE Born : DateType
  DECLARE Team : TeamType
ENDTYPE
```

- TYPE MemberType ... ENDTYPE
- a STRING field for the name
- fields of type DateType and TeamType

2.4 any two of:

- it can grow and shrink while the program runs, so the number of items need not be known when it is declared
- inserting or deleting in the middle changes only two pointers, instead of shuffling every later item along
- memory is used one node at a time, so none is reserved for items that are never stored

2.5

- NULL – there is no next node

3.1 C

- an enumerated type is a list of single named values, so it is **non-composite**
- a record, a set and a class each group several values together, so all three are composite

3.2

(a)

```
TYPE StatusType = (OnShelf, OnLoan, Lost)
```

- TYPE StatusType = (...)
- exactly the three values, and no others

(b)

```
TYPE BookType
    DECLARE Title : STRING
    DECLARE Author : STRING
    DECLARE YearPublished : INTEGER
    DECLARE Status : StatusType
ENDTYPE
```

- TYPE BookType ... ENDTYPE
- two STRING fields and an INTEGER field
- a field of type StatusType

(c)

```
DECLARE Catalogue : ARRAY[1:500] OF BookType
Catalogue[1].Status ← OnLoan
```

- the array declaration, of 500 elements of BookType
- the assignment, reaching the field with a dot and using the enumerated value unquoted

3.3

(a)

```
TYPE PNode = ^NodeType

TYPE NodeType
    DECLARE Value : INTEGER
    DECLARE NextPtr : PNode
ENDTYPE
```

- TYPE PNode = ^NodeType
- the pointer type declared **first**, because the record needs a type that already exists
- TYPE NodeType ... ENDTYPE with an INTEGER field
- a NextPtr field of type PNode

(b)

- inserting at the front makes the **most recent** value the first one, so HeadPointer points to the node holding **7**
- 20 was inserted first, so nothing was ever attached after it

- its `NextPtr` therefore still holds **NULL**, and it is the last node in the list

(c)

```
DECLARE Current : PNode
Current ← HeadPointer
WHILE Current <> NULL DO
    OUTPUT Current^.Value
    Current ← Current^.NextPtr
ENDWHILE
```

- start from the head: `Current ← HeadPointer`
- a loop that continues while `Current <> NULL`
- output the value and then move on with `Current ← Current^.NextPtr`, in that order
- $\triangle$ moving on before outputting loses the first value; testing `Current^.NextPtr <> NULL` instead loses the last one