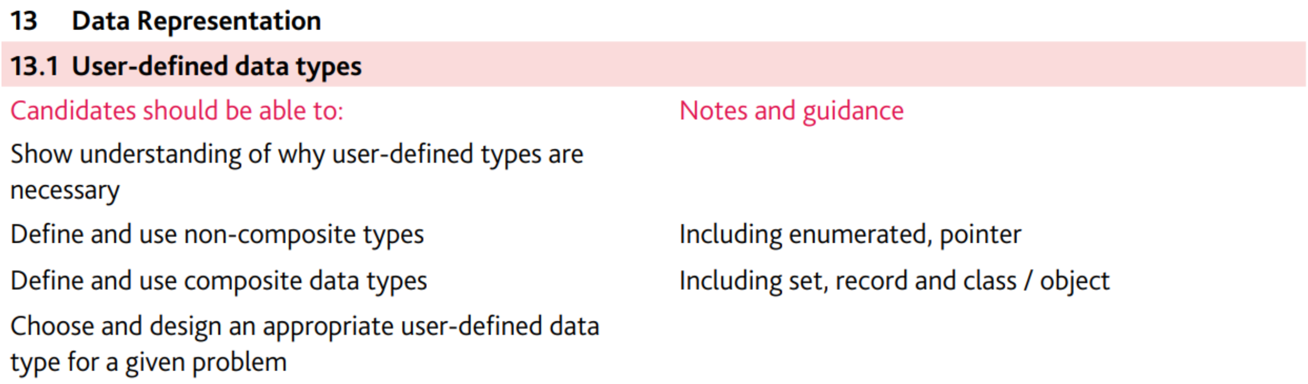
1. Non-composite Data Types



13 Data Representation

13.1 User-defined data types

Candidates should be able to:	Notes and guidance
Show understanding of why user-defined types are necessary	
Define and use non-composite types	Including enumerated, pointer
Define and use composite data types	Including set, record and class / object
Choose and design an appropriate user-defined data type for a given problem	

Purpose of user-defined data types:

- To create new data type (from existing data types)
- To allow data types not available in a programming language to be constructed // To extend the flexibility of programming language

Why user-defined data types are **necessary**:

- No suitable data type is provided by the language used
- The programmer needs to specify a new data type
- … that meets the **requirement** of the application

State what is meant by user-defined data types:

- Derived from one or more existing data types
- Used to extend the built-in data types
- Creates data types specific to applications

Non-composite Data Types

Define non-composite data types

- A single data type that does not involve a reference to another type / usually built in to a programming language

Give examples of non-composite data type:

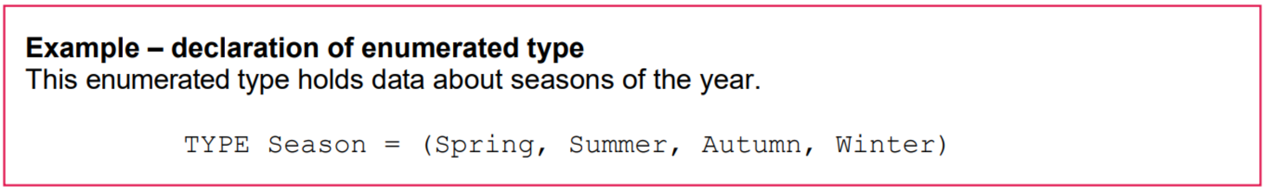
- Integer
  - Stores as a whole number
- Boolean
  - Stores true or false
- Real/double/float/decimal
  - Stores decimal numbers
- String
  - Stores zero or more characters
- Char
  - Stores a single character
- Pointer
  - Whole number used to reference a memory location

Enumerated

> This is a data type used to store constant values. It is a list of possible values.

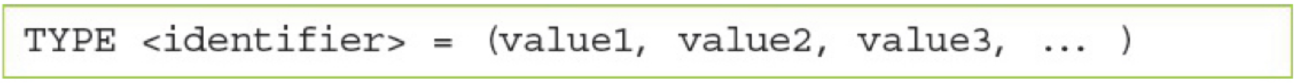
A user-defined non-composite data type with a list of possible values is called an enumerated data type. The enumerated type should be declared as follows:

```
TYPE <identifier> = (value1, value2, value3, ...)
```



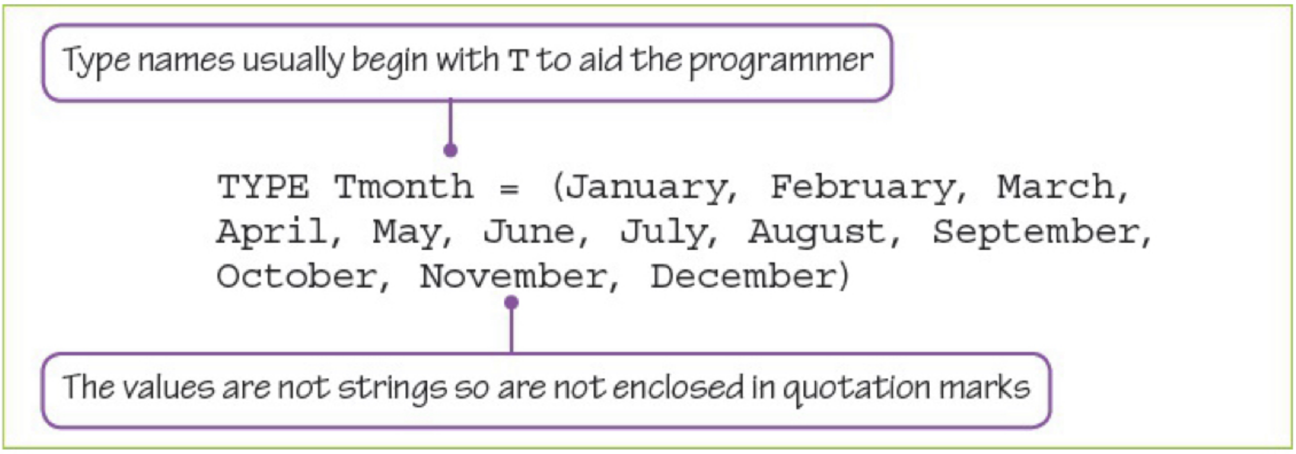
Example – declaration of enumerated type
This enumerated type holds data about seasons of the year.

```
TYPE Season = (Spring, Summer, Autumn, Winter)
```



```
TYPE <identifier> = (value1, value2, value3, ... )
```

For example, a data type for months of the year could be defined as:

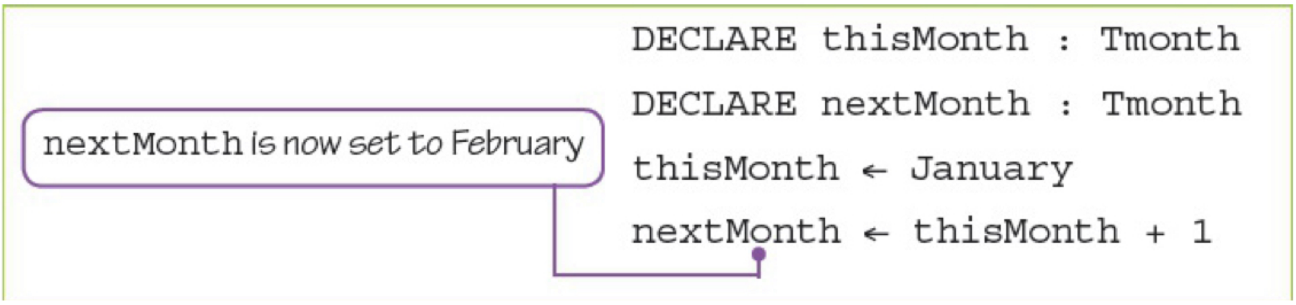


Type names usually begin with T to aid the programmer

```
TYPE Tmonth = (January, February, March,
April, May, June, July, August, September,
October, November, December)
```

The values are not strings so are not enclosed in quotation marks

Then the variables thisMonth and nextMonth of type Tmonth could be defined as:



```
DECLARE thisMonth : Tmonth
DECLARE nextMonth : Tmonth
thisMonth ← January
nextMonth ← thisMonth + 1
```

nextMonth is now set to February

```
TYPE SchoolDay = (Monday, Tuesday, Wednesday, Thursday,
                  Friday)
```

```
TYPE WeekEnd = (Saturday, Sunday)
```

⚠ : Enumerated 里面不用加引号 '' 或双引号 ""

⚠ : The values defined in an enumerated data type are ordinal. This mean that enumerated data types have an implied order or values.

Pointer ☝️

> A user-defined non-composite data type referencing a memory location is called a pointer

- □ Define: `TYPE <pointer> = ^<Typename>`
- □ Declaration: `DECLARE pointerVar : <pointer>`
- □ Reference: `pointerVar <- ^variable`
- □ Dereference: `pointerVar^`

The pointer should be defined as follows:

```
TYPE <pointer> = ^<Typename>
//For example:
TYPE IntPointer = ^INTEGER
DECLARE number: INTEGER
DECLARE numberLocation: IntPointer
numberLocation <- ^number //reference
```

Combining enumerated data type and pointer data type:

```
TYPE season = (Spring,Summer,Autumn,Winter)
TYPE seasonPointer = ^Season
DECLARE currentSeason: Season
currentSeason <- Spring
DECLARE mySeason: seasonPointer
mySeason <- ^currentSeason //reference
OUTPUT mySeason^ //dereference: output will be Spring
```