Exercise walk-through

User-defined data types ▪ 13.1

eXercise

You must be able to

- explain **why** user-defined types are needed
- define and use **enumerated** and **pointer** (non-composite) types
- define and use **set**, **record** and **class** (composite) types

Method — Why and the enumerated type

User-defined types make code **clearer** and let the compiler **reject illegal values**. An **enumerated** type is a fixed list of named constants:

```
TYPE Vehicle = (M100, M230, T101, T102, T120, T150)
DECLARE MyTaxi : Vehicle
MyTaxi ← T102
```

You cannot assign anything outside the list (the names are stored as small integers).

Method — Pointer type

A **pointer** holds the **memory address** of another variable (or `NULL`). `p^` **dereferences** it — reaches the variable it points to:

```
TYPE PNode = ^TNode
DECLARE p : PNode
p ← NEW TNode
p^.Value ← 42
```

Method — Pointer type

holds the address

p^ dereferences to reach the fields

TYPE Vehicle =

`M100` `M230` `T101` `T102` `T120` `T150`

`MyTaxi` may only hold one of these named values

An enumerated type is another user-defined type

Method — Composite types: set and class

A **set** is an unordered collection of unique values (operations: add, remove, membership, union):

```
DECLARE Available : SET OF Colour
Available ← {Red, Blue}
IF Green IN Available THEN OUTPUT "ok"
```

A **class** combines **attributes** (data) with **methods** (operations); an **object** is an instance of a class.

Practice 2.1 ▪ 1 mark

State **one** reason user-defined types are useful.

Solution 2.1

Method: Why and the enumerated type

Worked steps

Any one: makes code **self-documenting/clearer**; lets the compiler **reject illegal values**; groups related values that belong together B1.

Practice 2.2 ▪ 2 marks

Declare an **enumerated** type `Day` holding the seven days of the week.

Solution 2.2

Method: Pointer type

Worked steps

`TYPE Day = (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday)` M1 A1 *'TYPE … = (…)', seven named values.*

Practice 2.3 ▪ 1 mark

State what a **pointer** holds.

Solution 2.3

Method: Pointer type

Worked steps

The **memory address** of another variable (or `NULL` for no target) B1.

Practice 2.4 ▪ 1 mark

State what it means to **dereference** a pointer.

Solution 2.4

Worked steps

To **follow the pointer** and reach (read or change) the **variable it points to** B1.

Exam-style 3.1 ▪ 3 marks

Declare an enumerated type `Vehicle` with values `M100, M230, T101, T102`. Then declare a variable `MyTaxi` of that type and assign it `T101`.

Solution 3.1

> Method: Why and the enumerated type

Worked steps

```
TYPE Vehicle = (M100, M230, T101, T102)
DECLARE MyTaxi : Vehicle
MyTaxi ← T101
```

Solution 3.1

M1 A1 A1 *'TYPE ... = (...)' with the values, 'DECLARE MyTaxi : Vehicle', assignment of 'T101'*

Exam-style 3.2 ▪ 1 mark

(i) Declare a **pointer** type `PNode` that points to a `TNode`.
(ii) Write pseudocode to create a **new** `TNode` using a pointer `p` and set its `Value` to 5.

Solution 3.2

Method: Pointer type

Worked steps

(i) `TYPE PNode = ^TNode` B1.

3.2 (ii)

```
p ← NEW TNode
p^.Value ← 5
```

Solution 3.2

M1 A1 *'NEW TNode'*, *'p^.Value ← 5'* dereference + assign

Exam-style 3.3 ▪ 3 marks

A program records which of `{Red, Green, Blue}` are available. State why a **SET** is suitable, and write a line that tests whether `Green` is available.

Solution 3.3

Method: Composite types: set and class

Worked steps

A **set** stores **unordered, unique** values and supports a **membership test** — exactly what "is this colour available?" needs B1 B1. Test: `IF Green IN Available THEN …` B1.

Exam-style 3.4 ▪ 2 marks

State the difference between a **class** and an **object**.

Solution 3.4

Method: Composite types: set and class

Worked steps

A **class** is the **definition/template** (attributes + methods); an **object** is an **instance** of that class, holding its own data B1 B1.

Exam-style 3.5 ▪ 4 marks

A booking system stores one record for each activity choice. A user-defined type `Choice` holds a member's name, the activity chosen and the date.
(a) **Write** the pseudocode type declaration for `Choice`.
(b) **Write** the pseudocode statement that sets up a **single variable** `ThisChoice` of type `Choice`.
(c) **Write** the pseudocode statements that assign the name `"Ahmed"`, the activity `"Tennis"` and the date 12/03/2027 to that variable.
(d) A separate type `Activity` must hold the **names of up to 20 members** who have chosen one activity. **Write** its type declaration.
(e) `Choice` must now also record whether the booking has been **paid for**. **Write** the new statement required in the declaration, and **state** the data type you chose and why.
(f) **Explain** one advantage of a user-defined record type over holding the same values in three separate variables.

Solution 3.5

Worked steps

(a)

```
TYPE Choice
    DECLARE MemberName : STRING
    DECLARE ActivityName : STRING
    DECLARE ChoiceDate : DATE
ENDTYPE
```

Solution 3.5

B1 B1 B1 B1 *'TYPE'/'ENDTYPE'; all three fields; correct types; correct syntax throughout*

(b) `DECLARE ThisChoice : Choice` B1.

(c) `ThisChoice.MemberName ← "Ahmed"` B1; `ThisChoice.ActivityName ← "Tennis"` B1; `ThisChoice.ChoiceDate ← 12/03/2027` B1.

(d)

```
TYPE Activity
    DECLARE ActivityName : STRING
    DECLARE Members : ARRAY[1:20] OF STRING
ENDTYPE
```

Solution 3.5

B1 B1 B1 *'TYPE'/'ENDTYPE'; the array of 'STRING'; bounds '1:20'*

(e) `DECLARE Paid : BOOLEAN` inside the type B1. `BOOLEAN`, because the value can only be paid or not paid — two states, so a single bit of meaning, and no other value is valid B1.

(f) The three values belong to **one** booking, so keeping them in one record means they can be passed to a procedure, stored in an array or written to a file as a single item B1, and they cannot drift out of step with each other the way three separate variables can B1.