

Syllabus Content:

3.1 Data representation

3.1.1 User-defined data types

- why user-defined types are necessary, define and use non-composite types: enumerated, pointer
- define and use composite data types: set, record and class/object
- choose and design an appropriate user-defined data type for a given problem

Key terms

User-defined data type – a data type based on an existing data type or other data types that have been defined by a programmer

Non-composite data type – a data type that does not reference any other data types.

Enumerated data type – a non-composite data type defined by a given list of all possible values that has an implied order.

Pointer data type – a non-composite data type that uses the memory address of where the data is stored.

Set – a given list of unordered elements that can use set theory operations such as intersection and union.

User defined Data Type

You have already met a variety of built-in data types with integers, strings, chars and more. But often these limited data types aren't enough and a programmer wants to build their own data types. Just as an integer is restricted to "a whole number from - 2,147,483,648 through 2,147,483,647", user-defined data types have limits placed on their use by the programmer.

A **user defined data type** is a feature in most high level programming languages which allows a user (programmer) to define data type according to his/her own requirements

There are two categories of user defined data types.:

1. **Non-composite data type**
 - a. Enumerated data type
 - b. Pointer data type
2. **Composite**
 - a. Record data type
 - b. Set data type
 - c. Objects and classes

Non-composite user-defined data type:

A non-composite data type is defined without referencing another data type. They don't combine different built-in data types in one data type. Non-Composite data types are:

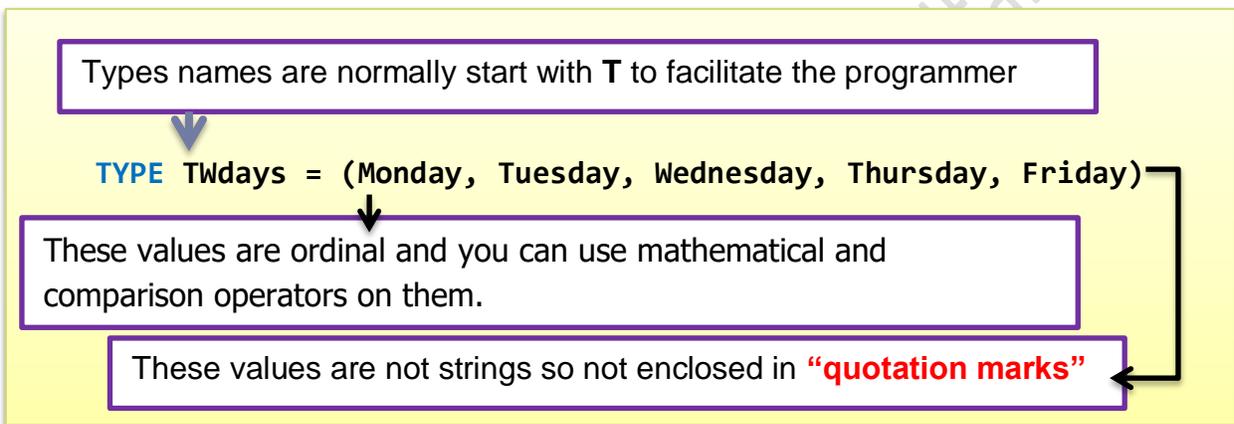
-  Enumerated data type
-  Pointers

Enumerated data type

An **enumerated data type** defines a list of possible values. The following pseudocode shows two examples of type definitions:

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

Pseudocode of data Type for **Working Days** can be declared as **TWdays**:



Variables can then be declared and assigned values, for example:

```

DECLARE Workdays : TWdays
DECLARE thisday : TWdays
DECLARE nextday : TWdays
today ← Monday
nextda ← today + 1

```

nextday is now set to Tuesday by applying mathematical operator +

It is important to note that the values of the enumerated type look like string values but they are not. They must not be enclosed in quote marks. This makes the second example much more useful because the ordering can be put to many uses in a program. For example, a comparison statement can be used with the values and variables of the enumerated data type:

```
DECLARE Weekend : Boolean
DECLARE day : TWdays
Weekend = TRUE IF day > Friday
```

KEY TERMS

Enumerated data type: a list of possible data values

Enumerated data type in vb.net

(Sample Program)

```
Module1.vb* -> X
ConsoleApplication9 -> Module1 -> Main
1  Module Module1
2  Enum Wdays ' Free Notes by Sir Majid Tahir
3      monday 'www.majidtahir.com
4      tuesday
5      wednesday
6      thursday
7      friday
8      saturday
9      sunday
10 End Enum
11
12 Sub Main()
13     Dim day As Wdays
14     Console.WriteLine("Please Enter day in a week")
15     day = Console.ReadLine()
16     If day > Wdays.friday Then
17         Console.WriteLine("Today is WeekEnd Buddy, Take a chill phill ")
18     Else
19         Console.WriteLine("Hurry up buddy, You are getting late for work, Today is " & day)
20     End If
21     Console.ReadLine()
22 End Sub
23
24 End Module
25
```

```
file:///c:/users/majid/documents/visual studio 2015/Projects/ConsoleApplica
Please Enter day in a week
7
Today is WeekEnd Buddy, Take a chill phill
```

Pointer Data Type:

A **pointer data type** is used to reference a memory location. This data type needs to have information about the type of data that will be stored in the memory location. In pseudocode the type definition has the following structure, in which **^** shows that the type being declared is a pointer and **<TYPENAME>** is the type of data to be found in the memory location: for example **INTEGER** or **REAL**: or any user-defined data type.

```
TYPE <pointer> = ^<Typename>
```

For example: a pointer for months of the year could be defined as follows:

```
TYPE TmonthPointer = ^Tmonth  
DECLARE monthPointer : TmonthPointer
```

Tmonth is the data type in the memory location that this pointer can be used to point to

It could then be used as follows:

```
monthPointer ← ^thisMonth
```

If the contents of the memory location are required rather than the address of the memory location: then the pointer can be **dereferenced**. For example: **myMonth** can be set to the value stored at the address **monthPointer** is pointing to:

```
DECLARE myMonth : Tmonth
```

```
myMonth ← monthPointer^
```

monthPointer has been dereferenced

Composite user-defined data types

A composite user-defined data type has a definition with reference to at least one other type. Composite data type can be a mixture of other built-in data types. Three examples are considered here.

Record data type

A **record data type** is the most useful and therefore most widely used. It allows the programmer to collect together values with different data types when these form a coherent whole.

KEY TERMS

Record data type: a data type that contains a fixed number of components, which can be of different types

As an example, a record could be used for a program using employee data. Pseudocode for defining the type could be:

```
TYPE TRecord
  DECLARE name      : STRING
  DECLARE Gender    : CHAR
  DECLARE DOB       : DATE
  DECLARE Regular   : BOOLEAN
  DECLARE Phone     : INTEGER
END TYPE
```

All type of data types are used in Record data type

A variable can then be declared as follows:

```
DECLARE newstudent : TRecord
```

individual data item can then be accessed using a dot notation:

```
newstudent.name ← "Muhammad"
```

A particular use of a record is for the implementation of a data structure where one or possibly two of the variables defined are pointer variables.

Record Data Type example

A Car Manufacturer and seller want to store details of cars. Details can be stored in Records

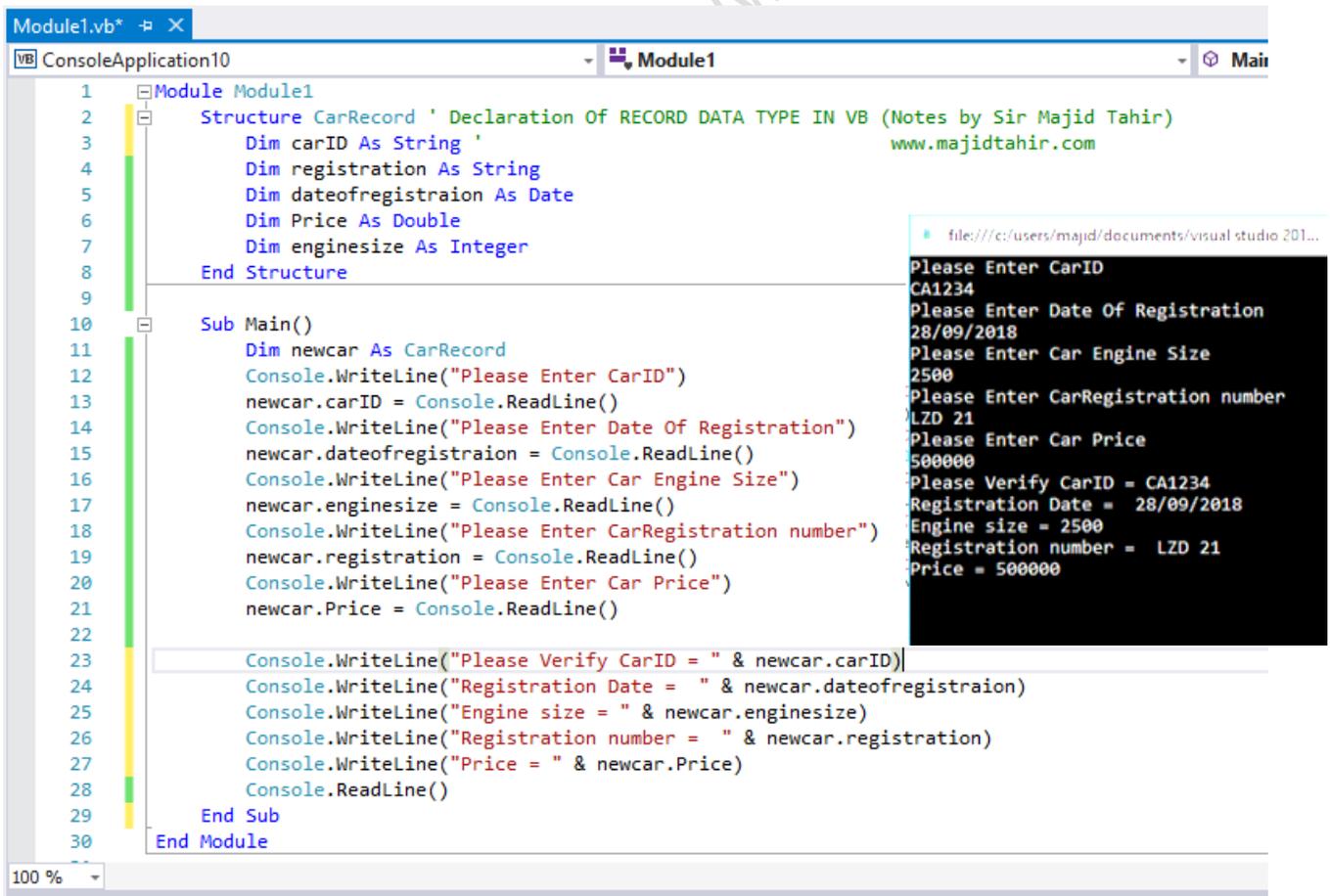
```
TYPE CarRecord
  DECLARE VehicleID      : STRING //UniqueItem & record key
  DECLARE Registrationnum : STRING
  DECLARE DateRegistration : DATE
  DECLARE Enginesize     : INTEGER
  DECLARE Purchaseprize  : REAL
END TYPE
```

Records in VB:

```
VB.NET Structure CarRecord
    Dim VehicleID As String
    Dim Registration As String
    Dim DateOfRegistration As Date
    Dim EngineSize As Integer
    Dim PurchasePrice As Decimal
End Structure
Dim ThisCar As CarRecord ' declare a variable of CarRecord type
Dim Car(100) As CarRecord ' declare an array of CarRecord type

ThisCar.EngineSize = 2500 ' assign value to a field
Car(2).EngineSize = 2500 ' assign value to a field of 2nd car in array
```

Sample code of Record Data Type in VB and its Output below:



```
Module1.vb* [X]
VB ConsoleApplication10 - Module1 - Main
1  Module Module1
2      Structure CarRecord ' Declaration of RECORD DATA TYPE IN VB (Notes by Sir Majid Tahir)
3          Dim carID As String ' www.majidtahir.com
4          Dim registration As String
5          Dim dateofregistraion As Date
6          Dim Price As Double
7          Dim enginesize As Integer
8      End Structure
9
10     Sub Main()
11         Dim newcar As CarRecord
12         Console.WriteLine("Please Enter CarID")
13         newcar.carID = Console.ReadLine()
14         Console.WriteLine("Please Enter Date Of Registration")
15         newcar.dateofregistraion = Console.ReadLine()
16         Console.WriteLine("Please Enter Car Engine Size")
17         newcar.enginesize = Console.ReadLine()
18         Console.WriteLine("Please Enter CarRegistration number")
19         newcar.registration = Console.ReadLine()
20         Console.WriteLine("Please Enter Car Price")
21         newcar.Price = Console.ReadLine()
22
23         Console.WriteLine("Please Verify CarID = " & newcar.carID)
24         Console.WriteLine("Registration Date = " & newcar.dateofregistraion)
25         Console.WriteLine("Engine size = " & newcar.enginesize)
26         Console.WriteLine("Registration number = " & newcar.registration)
27         Console.WriteLine("Price = " & newcar.Price)
28         Console.ReadLine()
29     End Sub
30 End Module
```

```
file:///c:/users/majid/documents/visual studio 201...
Please Enter CarID
CA1234
Please Enter Date Of Registration
28/09/2018
Please Enter Car Engine Size
2500
Please Enter CarRegistration number
LZD 21
Please Enter Car Price
500000
Please Verify CarID = CA1234
Registration Date = 28/09/2018
Engine size = 2500
Registration number = LZD 21
Price = 500000
```

Record Data type with Arrays in VB.net:

```
odule1.vb* X
Module1 Main
Module Module1
    'Notes of Sir Majid Tahir at www.majidtaahir.com
    Structure CarRecord 'Declaration Of Record Data Type
        Dim CarID As String
        Dim CarRegistration As String
        Dim DateOfRegistration As Date
        Dim CarManufacturer As String
        Dim EngineSize As Integer
        Dim SellingPrice As Double
    End Structure
    Sub Main()
        Dim newcar(5) As CarRecord '1 Dimension ARRAY of Record Data type declared
        For count = 1 To 5
            Console.WriteLine("Please Enter Car ID =" & count)
            newcar(count).CarID = Console.ReadLine()
            Console.WriteLine("Please Enter Car Registration = " & count)
            newcar(count).CarRegistration = Console.ReadLine()
            Console.WriteLine("Please Enter Car Manufacturer = " & count)
            newcar(count).CarManufacturer = Console.ReadLine()
            Console.WriteLine("Please Enter Car EngineSize = " & count)
            newcar(count).EngineSize = Console.ReadLine()
            Console.WriteLine("Please Enter Car Sale Price =" & count)
            newcar(count).SellingPrice = Console.ReadLine()
        Next
        For count = 1 To 5
            Console.WriteLine(count & " Car ID = " & newcar(count).CarID)
            Console.WriteLine(count & " Car Registration = " & newcar(count).CarRegistration)
            Console.WriteLine(count & " Date Of Registration = " & newcar(count).DateOfRegistration)
            Console.WriteLine(count & " Car Manufacturer = " & newcar(count).CarManufacturer)
            Console.WriteLine(count & " Car Engine size = " & newcar(count).EngineSize)
            Console.WriteLine(count & " Car Sale Price = " & newcar(count).SellingPrice)
        Next
        Console.ReadKey()
    End Sub
End Sub
```

Set data type:

A set data type allows a program to create sets and to apply the mathematical operations defined in set theory. The following is a representative list of the operations to be expected:

-  Union
-  Difference
-  Intersection
-  include an element in the set
-  exclude an element from the set
-  Check whether an element is in a set.

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed). However, the set itself is mutable. We

can add or remove items from it. The number of elements in a SET data type can vary, but no nulls are allowed.

Set can have any number of items and they may be of different types (integer, float, string etc.).

Assume we wish to create a **Set**. In PSEUDOCODE the type definition has this structure.

```
TYPE <set-identifier> = SET OF <Basetype>
```

The variable definition of set includes the elements of the set.

```
DECLARE <identifier> (value1, value 2, value3, ...):
```

A set of vowels can be declared as follows:

```
TYPE Sletter = SET OF CHAR  
DECLARE vowel ('a' , 'e' , 'i' , 'o' , 'u') : Sletter
```

Or we can declare sets in Pseudocodes this way
Here is another example

```
TYPE Sizes = SET OF String  
DECLARE SweaterSizes : (XXLarge, XLarge, Large, Medium, Small, XSmall)  
  
DECLARE Title: SET OF (Mr, Mrs, Ms, Miss, Dr)
```

NOTE: Of the three programming languages, **Pascal** and **Python** support the set data type.

The following Python code illustrates that the set data type has operators – union and intersection- which can be used in two sets:

How to create a set in PYTHON?

A set is created by placing all the items (elements) inside curly braces { }, separated by comma or by using the built-in function `set()`.

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like [list](#), [set](#) or [dictionary](#), as its element.

```
script.py IPython Shell
1 # set of integers
2 my_set = {1, 2, 3}
3 print(my_set)
4
5 # set of mixed datatypes
6 my_set = {1.0, "Hello", (1, 2, 3)}
7 print(my_set)
```

You can try creating SET and its operations online on below link:

<https://www.programiz.com/python-programming/set#operations>

Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods.

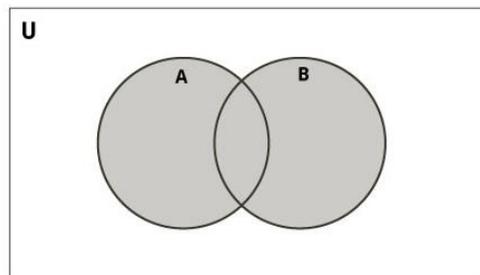
Let us consider the following two sets for the following operations.

1. >>> A = {1, 2, 3, 4, 5}
2. >>> B = {4, 5, 6, 7, 8}

```
script.py IPython Shell
1 # initialize A and B
2 A = {1, 2, 3, 4, 5}
3 B = {4, 5, 6, 7, 8}
4
5 # use | operator
6 # Output: {1, 2, 3, 4, 5, 6, 7, 8}
7 print(A | B)
```

Run

Set Union



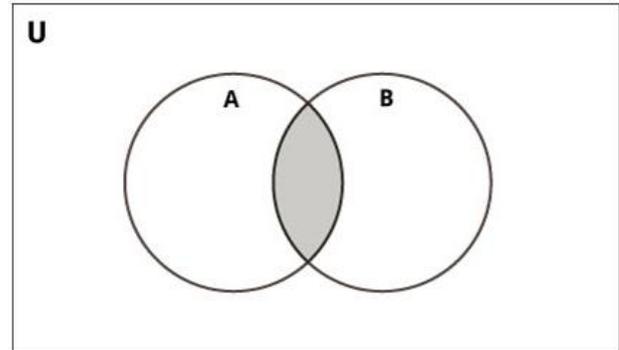
Union of **A** and **B** is a set of all elements from both sets.

Union is performed using **|** operator. Same can be accomplished using the method **union()**.

Set Intersection

```
script.py | Python Shell
1 # initialize A and B
2 A = {1, 2, 3, 4, 5}
3 B = {4, 5, 6, 7, 8}
4
5 # use & operator
6 # Output: {4, 5}
7 print(A & B)
```

Run



Intersection of A and B is a set of elements that are common in both sets. Intersection is performed using & operator. Same can be accomplished using the method `intersection()`.

Objects and classes

In object-oriented programming, a program defines the classes to be used - they are all user-defined data types. Then for each class the objects must be defined. Chapter 27 has a full discussion of this subject.

Why are user-defined data types necessary?

When object-oriented programming is not being used a programmer may choose not to use any user-defined data types. However, for any reasonably large program it is likely that their use will make a program more understandable and less error-prone. Once the programmer has decided because of this advantage to use a data type that is not one of the built-in types then user-definition is inevitable. The use of, for instance, an integer variable is the same for any program. However, there cannot be a built-in record type because each different problem will need an individual definition of a record.

References

- Book: AS and A-level Computer Science by Sylvia Langfield and Dave Duddell
- Book: AS and A-level Computer Science by David Watson & Hellen Williams
- <http://www.bbc.co.uk/education/guides/zjfgjxs/revision/1>
- https://www.tutorialspoint.com/vb.net/vb.net_constants.htm
- https://www.cs.drexel.edu/~introcs/F2K/lectures/5_Scientific/overflow.html
- <https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/underflow.html>