Chapter 13 Student Book Answers

13.1 What you should already know

1 a) String
b) Integer
c) Real
d) Date
e) Boolean

2
```
TYPE TAnimalRecord
    DECLARE Name : STRING
    DECLARE Species : STRING
    DECLARE DateOfBirth : DATE
    DECLARE Location : STRING
    DECLARE BornInZoo : BOOLEAN
    DECLARE Notes : STRING
ENDTYPE
```

Activity 13A

```
TYPE Tday = (Monday, Tuesday, Wednesday, Thursday, Friday,
Saturday, Sunday)
DECLARE today : Tday
DECLARE yesterday : Tday
today ← Wednesday
yesterday ← today - 1
```

Activity 13B

```
TYPE TdayPointer = ^Tday
DECLARE dayPointer : TdayPointer
dayPointer ← ^today
```

Activity 13C

1 A composite data type refers to other data types in its definition. A non-composite data type does not refer to other data types.

2 Use defined data types allow programs to be more readable for other programmers. For example, using the days of the week as an enumerated data type.

3 a) An enumerated data type, as a list of colours can be provided with meaningful names used for each colour in the list.

b) A record structure that contains different types of data would be used, so the data for each house can be used together in one structure.

c) pointer data type as this will reference the address/location of the integer stored in main memory.

13.2 What you should already know

1 Read, to read the data stored in the file.
Write to write data to a file, this will overwrite any data stored in the file.
Append to add new data to the end of a file.

2 **a)**
```
DECLARE myTextFile : STRING
myTextFile ← "myText.txt"
```

b)
```
DECLARE textLn : STRING
OPEN myTextFile FOR WRITE
REPEAT
    OUTPUT "Please enter a line of text"
    INPUT textLn
    IF textLn <> ""
      THEN
        WRITEFILE, textLn
      ELSE
        CLOSEFILE(myTextFile)
    ENDIF
UNTIL textLn = ""
```

c)
```
OUTPUT "The file contains these lines of text:"
OPEN myTextFile FOR READ
REPEAT
    READFILE, textLn
    OUTPUT textLn
UNTIL EOF(myTextFile)
CLOSEFILE(myTextFile)
```

d)
```
DECLARE textLn : STRING
OPEN myTextFile FOR APPEND
REPEAT
    OUTPUT "Please enter a line of text"
    INPUT textLn
    IF textLn <> ""
      THEN
        WRITEFILE, textLn
      ELSE
        CLOSEFILE(myTextFile)
    ENDIF
UNTIL textLn = ""
```

3

Python

```
# writing to a file, reading a line of text from a file,
appending a line of text
myFile = open ("myText.txt","w")
textLn = "start"
while textLn != "":
    textLn = input("Please enter a line of text ")
    if textLn != "":
        myFile.write(textLn + "\n")
    else: myFile.close()

print("The file contains these lines of text")
myFile = open ("myText.txt","r")
textLn = myFile.read()
print(textLn)

myFile = open ("myText.txt","a")
textLn = "append"
while textLn != "":
    textLn = input("Please enter a line of text ")
    if textLn != "":
        myFile.write(textLn + "\n")
    else: myFile.close()

print("The file now contains these lines of text")
myFile = open ("myText.txt","r")
textLn = myFile.read()
print(textLn)
```

VB

```
'writing to, reading from and appending to a text file
Imports System.IO

Module Module1

    Sub Main()
        Dim textLn As String
        Dim objMyFileWrite As StreamWriter
        Dim objMyFileAppend As StreamWriter
        Dim objMyFileRead As StreamReader
        Dim objMyFileReadAgain As StreamReader

        objMyFileWrite = New StreamWriter("textFile.txt")
        Console.Write("Please enter a line of text ")
        textLn = Console.ReadLine()

        Do
            objMyFileWrite.WriteLine(textLn)
            Console.Write("Please enter a line of text ")
            textLn = Console.ReadLine()
        Loop Until textLn = ""

        objMyFileWrite.Close()

        objMyFileRead = New StreamReader("textFile.txt")
        Do While Not textLn Is Nothing
            textLn = objMyFileRead.ReadLine
            Console.WriteLine(textLn)
        Loop
        objMyFileRead.Close()

        objMyFileAppend = New StreamWriter("textFile.txt", True)
        Console.Write("Please enter a line of text ")
        textLn = Console.ReadLine()
        Do
            objMyFileAppend.WriteLine(textLn)
            Console.Write("Please enter a line of text ")
            textLn = Console.ReadLine()
        Loop Until textLn = ""
        objMyFileAppend.Close()

        objMyFileReadAgain = New StreamReader("textFile.txt")
        Do While Not textLn Is Nothing
            textLn = objMyFileReadAgain.ReadLine
            Console.WriteLine(textLn)
        Loop


        objMyFileReadAgain.Close()

        Console.ReadLine()

    End Sub

End Module
```

Java

```
//writing to and reading from a text file
import java.util.Scanner;
import java.io.BufferedReader;
import java.io.PrintWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

class WhatYouNeedToKnow13_2 {

    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);
        String textLn;
        try {
            FileWriter myFileWriter = new FileWriter("textFile.txt", false);
            PrintWriter myPrintWriter = new PrintWriter(myFileWriter);
            System.out.println("Please enter a line of text  ");
            textLn = myObj.next();
            do {
                myPrintWriter.printf("%s" + "%n", textLn);
                System.out.println("Please enter a line of text  ");
                textLn = myObj.next();
                }
            while (!textLn.equals("end"));
            myPrintWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            FileReader myFileReader  = new FileReader("textFile.txt");
            BufferedReader myBufferReader  = new BufferedReader(myFileReader);
            textLn = myBufferReader.readLine();
            do {
                System.out.println(textLn);
                textLn = myBufferReader.readLine();
            }
            while(textLn != null);
            myFileReader.close();

        } catch (IOException e) {
            e.printStackTrace();
        }

try {
            FileWriter myFileWriter = new FileWriter("textFile.txt", true);
            PrintWriter myPrintWriter = new PrintWriter(myFileWriter);
            System.out.println("Please enter a line of text  ");
            textLn = myObj.next();
            do {
                myPrintWriter.printf("%s" + "%n", textLn);
                System.out.println("Please enter a line of text  ");
                textLn = myObj.next();
                }
            while (!textLn.equals("end"));
            myPrintWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        try {
            FileReader myFileReader  = new FileReader("textFile.txt");
```

```
                BufferedReader myBufferReader  = new BufferedReader(myFileReader);
                textLn = myBufferReader.readLine();
                do {
                    System.out.println(textLn);
                    textLn = myBufferReader.readLine();
                }
                while(textLn != null);
                myFileReader.close();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }

    }
```

Activity 13D

As 9354 ÷ 1000 is 9 remainder is 354, with 5 locations for each record, this record would be stored at address 2175 = 500 + 353 ∗5 and the next four locations, assuming that the first record is stored at address 500.

Activity 13E

1 Serial file each new record is added to the end of a file, for example a log of temperature readings taken at a weather station.
Sequential file records are stored in a given order, usually based on the key field, for example ascending number of employee number for a personnel file.

2 See section 13.2.1

3 **a)** Random access as only one record is required at a time, low hit rate.

b) Sequential access as all the records need to be accessed, high hit rate.

c) Serial access, as each record is added to the end of the file in chronological order.

13.3 What you should already know

1 **a)** 00110000
b) 01111010
c) 10011100
d) 11001001
e) 11111110

2 **a)** 51
b) 126
c) −77
d) −14
e) −1

3 **a)** 11001100
b) 11100011
c) 10110100
d) 11000001
e) 10000010

4 **a)** 01001111
b) 10000000
c) 10000001
d) 11011001
e) 1)01001100
f) 1)00100010
g) 1)00000001
h) 11111100
i) 1)11011111
j) 11101011

5 **a)** 1.23×10^{8}
b) 2.505×10^{15}
c) -1.2×10^{3}
d) 2.341×10^{-9}
e) 1.24005×10^{-5}

6 **a)** **i)** $2.1/5 \times 10^{1}$
ii) $1.17/4 \times 10^{2}$
iii) $5.58/20 \times 10^{2}$

b) **i)** $11/16 \times 8\ (2^{3})$
ii) $41/64 \times 16\ (2^{4})$
iii) $52/64 \times 16\ (2^{4})$

Activity 13F

a) $39/64 \times 2^5 = 19.5$

b) $41/128 \times 2^7 = 41$

c) $7/8 \times 2^{-5} = 7/256$ (0.02734375)

d) $15/64 \times 2^{-4} = 15/1024$ (0.0146484375)

e) $7/8 \times 2^3 = 7$

f) $-13/16 \times 2^2 = -3.25$

g) $-3/32 \times 2^4 = -1.5$

h) $-5/8 \times 2^5 = -20$

i) $-5/8 \times 2^{-3} = -5/64$ (−0.078125)

j) $-1/4 \times 2^{-6} = -1/256$ (−0.00390625)

Activity 13G

1a)

0	0	1	0	1	1	0	0	0	0	0	0	0	1	1	1

b)

0	0	1	0	0	1	1	0	0	0	0	0	0	0	1	1

c)

0	0	0	1	0	1	1	0	1	1	1	1	1	0	1	1

d)

0	1	1	1	1	0	0	0	1	1	1	1	1	1	0	1

e)

0	1	0	1	0	1	0	0	0	0	0	0	0	1	0	1

f)

1	1	1	0	1	0	1	0	0	0	0	0	0	1	0	0

g)

1	0	1	1	1	0	0	0	1	1	1	1	1	1	1	1

h)

1	1	0	1	1	0	0	0	0	0	0	0	0	1	0	1

i)

1	1	1	0	0	0	0	0	1	1	1	1	1	0	1	0

..j)

1	0	1	1	0	0	0	0

1	1	1	1	1	1	1	0

2 **a)** $3.5 = 7/2 = 7/8 \times 2^2 = 0.1110000 \times 00000010$

b) $0.3125 = 10/32 = 5/16 = 0.0101000 \times 00000000$ (or $0.1010000 \times 11111111$)

c) $15.375 = 123/8 = 123/128 \times 2^4 = 0.1111011 \times 00000100$

d) $41/64 = 0.1010010 \times 00000000$

e) $9.125 = 73/8 = 73/128 \times 2^4 = 0.1001001 \times 00000100$

f) $-15/32 = -1 + 17/32 = 1.1000100 \times 00000000$

g) $-3.5 = -7/2 = -7/8 \times 2^2 = 1.0010000 \times 00000010$

h) $-10.25 = -41/4 = -41/64 \times 2^4 = 1.0101110 \times 00000100$

i) $-67/64 = -67/128 \times 2^1 = 1.0111111 \times 00000001$

j) $-107/32 = -107/128 \times 2^2 = 1.0010101 \times 00000010$

Activity 13H

a) 0.1101000 × 00000011

b) 0.1100000 × 00000111

c) 0.1110000 × 00000010

d) 0.1000100 × 00000001

e) 0.1110000 × 00000110

f) 1.0000000 × 00000100

g) 1.0010000 × 00001010

h) 1.0110000 × 00000000

i) 0.1111000 × 11110101

j) 1.0000000 × 11110000

Activity 13I

1 Largest:

0	1	1	1	1	1	1	1	1	1

0	1	1	1	1	1

Smallest magnitude:

0	1	0	0	0	0	0	0	0	0

1	0	0	0	0	0

2 Accuracy (precision) is increased by increasing size of mantissa.

Range is increased by increasing size of component.

3 **a)** This will cause an overflow error since the number > maximum number which can be stored.

b) underflow error will occur since division by zero generates a very small number < smallest value which can be stored.

4 a) .88 × 2 = **1**.76 so we will use the **1** value to give 0**.1**

.76 × 2 = **1**.52 so we will use the **1** value to give 0**.11**

.52 × 2 = **1**.04 so we will use the **1** value to give 0**.111**

.04 × 2 = **0**.08 so we will use the **0** value to give 0**.1110**

.08 × 2 = **0**.16 so we will use the **0** value to give 0**.11100**

.16 × 2 = **0**.32 so we will use the **0** value to give 0**.111000**

.32 × 2 = **0**.64 so we will use the **0** value to give 0**.1110000**

.64 × 2 = **1**.28 so we will use the **1** value to give 0**.11100001**

.28 × 2 = **0**.56 so we will use the **0** value to give 0**.111000010**

.56 × 2 = **1**.12 so we will use the **1** value to give 0**.1110000101**

We have to stop here since our system uses a maximum of 10 bits. Now the value of 2 (in binary) is 0010; this therefore gives us:

$$2.88 = 0010.1110000101$$

Moving the binary point as far to the left as we can gives us:

$0.101110000101 \times 2^2$ (2^2 since we moved the point 3 places)

Thus, we get 0.101110000 × 000010

(mantissa) (exponent)

This is equal to: $23/32 \times 2^2 = 23/8 =$ **2.875**

So, 2.88 is stored as 2.875 in our floating-point system.

b) .38 × 2 = **0**.76 so we will use the **0** value to give 0**.0**

.76 × 2 = **1**.52 so we will use the **1** value to give 0**.01**

.52 × 2 = **1**.04 so we will use the **1** value to give 0**.011**

.04 × 2 = **0**.08 so we will use the **0** value to give 0**.0110**

.08 × 2 = **0**.16 so we will use the **0** value to give 0**.01100**

.16 × 2 = **0**.32 so we will use the **0** value to give 0**.011000**

.32 × 2 = **0**.64 so we will use the **0** value to give 0**.0110000**

.64 × 2 = **1**.28 so we will use the **1** value to give 0**.01100001**

.28 × 2 = **0**.56 so we will use the **0** value to give 0**.011000010**

We have to stop here since our system uses a maximum of 10 bits. Now the value of −5 (in binary) is −0101; this therefore gives us:

$$-5.88 = -0101.011000010$$

Moving the binary point as far to the left as we can gives us:

-0.101011000×2^3 (2^3 since we moved the point 3 places)

Thus we get − 0.101011000 000011

(mantissa) (exponent)

Applying two's complement we get:

(1.010100111 + 1) × 000011

i.e. 1.010101000 × 000011

This is equal to: $-43/64 \times 2^3 = -43/8 =$ **−5.375**

So − 5.38 is stored as − 5.375 in our floating-point system.

End of chapter questions

1 a) i) A = −1 + ¼ × 2^{-1} = −3/4 × 1/2 = −3/8 = −0.375

B = (1/2 + 1/8) × 2^3 = 5/8 × 8 = 5

C = (1/8 + 1/32) × 2^5 = 5/32 × 32 = 5

ii) C

iii) Because there could be more than one way to represent the same value and some options are not possible (e.g. 0.1000000 × 00000010 could become 0.0000000 × 00001001 which is not possible).

b) **Accuracy and range**

- 16-bit mantissa and 8-bit exponent gives high accuracy but small range.
- Increasing mantissa size would increase accuracy further ….
- …. but correspondingly reducing the exponent size would reduce the range (and the converse is true).

c) Because normalised values must have 1.0 or 0.1 therefore it is not possible to store the value zero using this method.

2 a) i) (1/2 + ¼ + 1/8 + 1/64) × 2^7 = 57/64 × 2^7 = 114

ii) (−1 + ¼ + 1/32 + 1/64 + 1/128) × 2^{-4} = −89/128 × 1/16 = −89/2^{11} = 0.000434571

b) i) 4.75 = 19/4 = 19/32 × 2^3 = 0.100110000000000 × 000011

ii) −8.375 = −67/8 = −67/128 × 2^4 = (−1 + 61/128) × 2^4

= 1.011110100000000 × 000100

3 a) +3.5

01110000 00000010

= 11.1

= 0.11 × 2^2

b) −3.5

10010000 00000010

One's complement of 8-bit mantissa for +3.5 gives:

10010000

4 a) i) `Tseason`

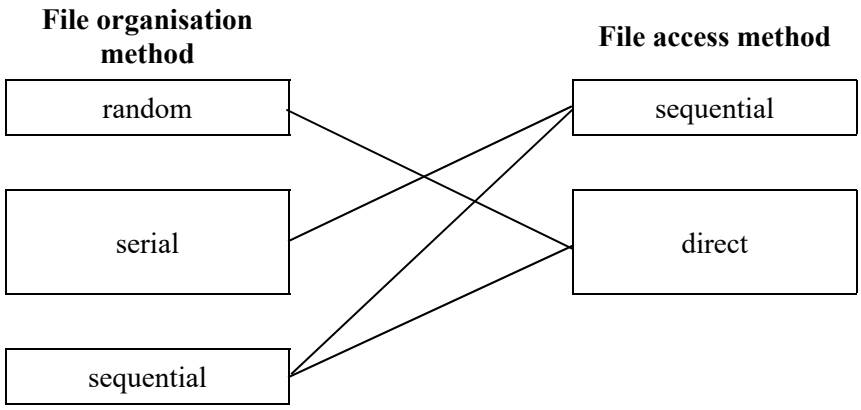
ii) `TJournalRecord`

iii) `STRING`

vi) `TJournalRecord` or `Tseason`

b)
```
DECLARE Journal : TJournal
  Journal.title ← "Spring Flowers"
  Journal.author ← "H Williams"
  Journal.publisher ← "XYZ Press"
  Journal.noPages ← 40
  Journal.season ← Spring
```

5 a)



b) A – serial, as meter readings are added to the end of file and stored chronologically.
B – sequential as each customer has a unique account number and the file is sorted on the account number and there is a high hit rate.
C – random organisation allows fastest direct access to the required record, so this is suitable for access to individual records.