Week 39

A-Level Computer Science

Homework bundle for this week

本周作业合集

exercises.lol

Contents 目录

A-Level Computer Science Name: _____ Score: _____

1. Why are user-defined types needed? Select **all** that apply.

   ☐ they restrict a field to its legal values, so the compiler catches more mistakes

   ☐ they group values of different types that describe one thing

   ☐ the declaration says what the variable is, not just how it is stored

   ☐ they make the program run faster

   Tick every correct option.

2. A serial file stores records:

   ☐ in the order they were added, with no sorting

   ☐ sorted by a key field

   ☐ at positions computed from a key

   ☐ in random memory addresses

3. A hash function:

   ☐ turns a record's key into a storage address

   ☐ sorts the file by key

   ☐ encrypts the record

   ☐ compresses the data

4. A floating-point number is stored as:

   ☐ a mantissa multiplied by 2 raised to an exponent

   ☐ a single integer

   ☐ a string of digits

   ☐ a denary fraction

5. An enumerated type holds:

   ☐ a value from a fixed list of named constants

   ☐ any string at all

   ☐ a memory address

   ☐ a group of different fields

6. A random (direct-access) file places each record:

   ☐ at a position computed from its key, giv ery fast direct lookup

☐ in arrival order

☐ in alphabetical order only

☐ always at the end

7. What makes a hash function a good one? Select **all** that apply.

   ☐ it is fast to compute

   ☐ the same key always gives the same address

   ☐ it spreads keys evenly over the slots

   ☐ it never produces a collision

   Tick every correct option.

8. A pointer stores the memory address of another variable, and dereferencing it (e.g. ‘p^‘) reaches the variable it points to.

   ☐ True ☐ False

9. Using the modulo hash ‘address ← key MOD N‘ with key = 27 and N = 10, what address is produced?

   _____

10. Written as a normalised fraction times a power of two, 2.5 = 0.101 x 2^___. Give the exponent.

    _____

13. Data Representation

Answers · 答案

A-Level Computer Science

1. **they restrict a field to its legal values, so the compiler catches more mistakes; they group values of different types that describe one thing; the declaration says what the variable is, not just how it is stored**

   Restriction, grouping and self-documentation. Speed is not the reason; correctness and clarity are.

2. **in the order they were added, with no sorting**

   Serial files keep records in arrival order — fast to append, slow to search. Sequential files are sorted by key.

3. **turns a record’s key into a storage address**

   A hash function maps a key to an address, enabling near-instant direct lookup.

4. **a mantissa multiplied by 2 raised to an exponent**

   value = mantissa × 2^exponent — binary scientific notation, with both parts stored in two’s complement.

5. **a value from a fixed list of named constants**

   An enumerated type restricts a variable to one of a fixed set of named values (e.g. days of the week).

6. **at a position computed from its key, giving very fast direct lookup**

   A key is converted (often by a hash) to a position, so a single-key lookup is very fast.

7. **it is fast to compute; the same key always gives the same address; it spreads keys evenly over the slots**

   Fast, deterministic and evenly spread. No realistic hash avoids collisions entirely, which is why every design includes a resolution strategy.

8. **True**

   Pointers (with NULL marking ”points to nothing”) are how dynamic structures like linked lists and trees are built.

9. **7**

   27 MOD 10 = 7 (the remainder when 27 is divided by 10).

10. **2**

    2.5 is 10.1 in binary; sliding the point two places left gives 0.101, so the exponent is 2 and the mantissa is 01010000.

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

13.1 User-defined data types

Name: _____ Class: _____ Date: _____ **Total: 31 marks**

KEY WORDS class 类 • object 对象 • record 记录 • enumerated type 枚举类型 • user-defined data types 用户定义类型

Objective

Build the skills to answer exam questions on **user-defined data types** 用户定义类型 — enumerated, pointer, set, record and class.

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

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

■ Why and the enumerated type

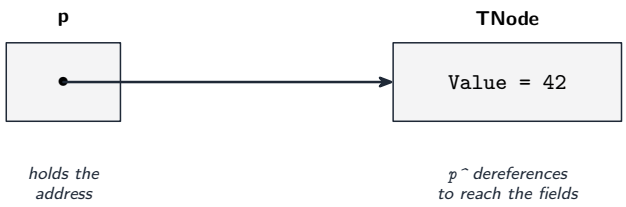
User-defined types make code **clearer** and let the compiler **reject illegal values**. An **enumerated** type is a fixed list of named constants:

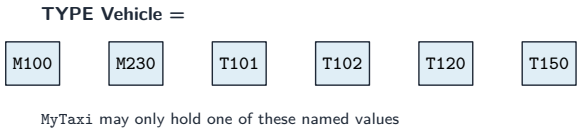
```
TYPE Vehicle = (M100, M230, T101, T102, T120, T150)
DECLARE MyTaxi : Vehicle
MyTaxi ← T102
```

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

■ Pointer type

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





An enumerated type is another user-defined type

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

■ 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.

2 Practice

Now apply the methods above.

2.1 State **one** reason user-defined types are useful. [1]

2.2 Declare an **enumerated** type `Day` holding the seven days of the week. [2]

2.3 State what a **pointer** holds. [1]

2.4 State what it means to **dereference** a pointer. [1]

3 Exam-style questions

3.1 Declare an enumerated type `Vehicle` with values `M100, M230, T101, T102`. Then declare a variable `MyTaxi` of that type and assign it `T101`. [3]

3.2 (i) Declare a **pointer** type `PNode` that points to a `TNode`. [1]

(ii) Write pseudocode to create a **new** `TNode` using a pointer `p` and set its `Value` to `5`. [2]

3.3 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. [3]

3.4 State the difference between a **class** and an **object**. [2]

3.5 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`. [4]

(b) **Write** the pseudocode statement that sets up a **single variable** `ThisChoice` of type `Choice`. [1]

(c) **Write** the pseudocode statements that assign the name `"Ahmed"`, the activity `"Tennis"` and the date `12/03/2027` to that variable. [3]

(d) A separate type `Activity` must hold the **names of up to 20 members** who have chosen one activity. **Write** its type declaration. [3]

(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. [2]

(f) **Explain** one advantage of a user-defined record type over holding the same values in three separate variables. [2]

Solutions

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

2.1

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

2.2

- `TYPE … = ( … )` with the seven named values
- `TYPE Day = (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday)`

2.3

- the **memory address** of another variable (or `NULL` for no target)

2.4

- to **follow the pointer** and reach (read or change) the **variable it points to**

3.1

- `TYPE … = (…)` with the values, then declare and assign

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

3.2

(i) `TYPE PNode = ^TNode`

(ii) `NEW` then dereference and assign

```
p ← NEW TNode
p^.Value ← 5
```

3.3

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

3.4

- a **class** is the **definition/template** (attributes + methods)
- an **object** is an **instance** of that class, holding its own data

3.5

(a)

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

- TYPE/ENDTYPE; all three fields; correct types; correct syntax

(b) `DECLARE ThisChoice : Choice`

(c) `ThisChoice.MemberName ← "Ahmed"`

- `ThisChoice.ActivityName ← "Tennis"`
- `ThisChoice.ChoiceDate ← 12/03/2027`

(d)

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

- TYPE/ENDTYPE; the array of `STRING`; bounds `1:20`

(e) `DECLARE Paid : BOOLEAN` inside the type

- `BOOLEAN`, because the value can only be paid or not paid — two states, and no other value is valid

(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

- they cannot drift out of step with each other the way three separate variables can

1 The composite record data type, `ClubMember`, is defined in pseudocode as:

```
TYPE ClubMember
  DECLARE Code : INTEGER
  DECLARE LastName : STRING
  DECLARE FirstName : STRING
  DECLARE Telephone : STRING
  DECLARE JoinDate : DATE
  DECLARE Fees : REAL
  DECLARE FeesPaid : BOOLEAN
ENDTYPE
```

(a) (i) Write the **pseudocode** statement to set up a variable for one record of the composite data type, `ClubMember`.

.....

..... [1]

(ii) Write the **pseudocode** statements to assign the following values to the variable set up in part **(a)(i)**:

- `984632` to `Code`
- `TRUE` to `FeesPaid`

.....

.....

.....

..... [2]

(b) An enumerated data type, `Activity`, is required, so that a new field, `Choice`, can be added to the composite data type, `ClubMember`, to allow members to choose an activity.

(i) Write the **pseudocode** statement for the type declaration of `Activity` to hold the names of the available activities:

Badminton, Football, Golf, Snooker, Swimming, Tennis.

.....

.....

.....

..... [2]

(ii) Write the **new pseudocode** statement required to update the declaration of `Choice` in the definition of `ClubMember`.

....

..... [1]

1 **(a)** An enumerated data type, `Spectrum`, is required to hold the names of colours.

(i) Write the **pseudocode** statement for the type declaration of `Spectrum` to hold the names of the colours available:

Red, Orange, Yellow, Green, Blue, Indigo, Violet.

.....

..... [2]

(ii) Identify **two** characteristics of the list of values given in an enumerated data type declaration.

1 .....

.....

2 .....

..... [2]

(b) `ColourData` is a composite data type to store definitions of colours.

Write **pseudocode** statements to declare `ColourData` to hold the following fields:

Field	Example data
ColourCode	`XYZ12345`
Colour	`Red`
Wavelength	`650`
Frequency	`4.62`
PrimaryColour	`Yes`

Use the most appropriate data type in each case, including the enumerated data type `Spectrum` from part **a(i)**.

.....

.....

.....

.....

.....

.....

.....

....

..... [4]

2 A program stores data about trains and train stations using Object-Oriented Programming (OOP).

The class `Train` stores the data about the trains:

Train	
`TrainIDNumber : String`	stores the train ID number
`Route : Integer`	stores the route number the train is travelling
`Constructor()`	initialises `TrainIDNumber` and `Route` to the parameter values
`GetTrainIDNumber()`	returns the train ID number
`GetRoute()`	returns the route number the train is travelling

(a) **(i)** Write program code to declare the class `Train` and its constructor.

Do **not** declare the other methods.

All attributes should be private.

Use your programming language appropriate constructor.

If you are writing in Python, include attribute declarations using comments.

Save your program as **Question2_N25**.

Copy and paste the program code into part **2(a)(i)** in the evidence document.

[4]

(ii) The methods `GetTrainIDNumber()` and `GetRoute()` return the appropriate attribute.

Write program code for `GetTrainIDNumber()` and `GetRoute()`

Save your program.

Copy and paste the program code into part **2(a)(ii)** in the evidence document.

[3]

(b) The program is tested with four trains:

train ID number	route
12ADV	134
33ART	20
9FKF	3
21VBC	24

Write program code to declare an instance of `Train` for each of the **four** trains.

Save your program.

Copy and paste the program code into part **2(b)** in the evidence document.

[2]

(c) The class `Station` stores the data about the stations:

Station	
`StationID : String`	stores the station ID
`NumberPlatforms : Integer`	stores the number of platforms at the station
`Trains[0:9] : Train`	stores the trains currently at the station platforms
`NumberTrains : Integer`	stores the number of trains currently at the station platforms
`Constructor()`	initialises `StationID` and `NumberPlatforms` to the parameter values, initialises `Trains` to an empty array and `NumberTrains` to `0`
`GetTrains()`	returns a string containing data about the trains currently at the station platforms
`AddTrain()`	takes a `Train` parameter and stores it if there is a platform available; each platform can only have one train

(i) Write program code to declare the class `Station` and its constructor.

Do **not** declare the other methods.

All attributes should be private.

Use your programming language appropriate constructor.

If you are writing in Python, include attribute declarations using comments.

Save your program.

Copy and paste the program code into part **2(c)(i)** in the evidence document.

[3]

(ii) The method `AddTrain()` takes a `Train` parameter. The method compares the attributes `NumberTrains` and `NumberPlatforms` to identify if there is a platform available (a platform currently with no train).

The method returns `FALSE` if there are no platforms available.

If there is a platform available, the method:

- stores the `Train` parameter in the array `Trains`
- updates the appropriate attribute(s)
- returns `TRUE`

Write program code for `AddTrain()`

Save your program.

Copy and paste the program code into part **2(c)(ii)** in the evidence document.

[4]

(iii) The method `GetTrains()` returns the string `"There are no trains"` if there are no trains at the station platforms.

If there are trains at the station platforms, the method returns a string in the format:

```
The trains at station <StationID> are:
<TrainIDNumber> on route number <Route>
```

The line `<TrainIDNumber> on route number <Route>` is repeated for each train at the station platforms.

For example: If the station with the station ID `"NT1"` has two trains with the train ID numbers `"48RTG"`, `"6UFH"`, the method will produce this output:

```
The trains at station NT1 are:
48RTG on route number 43
6UFH on route number 12
```

Write program code for `GetTrains()`

Save your program.

Copy and paste the program code into part **2(c)(iii)** in the evidence document.

[6]

(d) The program is tested with two stations:

station ID	number of platforms
STH	2
NTH	1

(i) Write program code to amend the main program to declare an instance of `Station` for each of the **two** stations.

Save your program.

Copy and paste the program code into part **2(d)(i)** in the evidence document.

[2]

(ii) The four trains attempt to stop at the following stations in the order given:

- Train 12ADV, station STH
- Train 33ART, station STH
- Train 9FKF, station STH
- Train 21VBC, station NTH

Write program code to amend the main program to:

- add each train to the given station using `AddTrain()`
- output `"Station is full"` for any train where the return value indicates it cannot be added to the station
- output the trains at each station using `GetTrains()`

Save your program.

Copy and paste the program code into part **2(d)(ii)** in the evidence document.

[4]

(iii) Test your program.

Take a screenshot of the output(s).

Save your program.

Copy and paste the screenshot(s) into part **2(d)(iii)** in the evidence document.

[2]

1 A programmer is writing a program to manage bookings for a small taxi company. The programmer requires some user-defined data types.

(a) Write a **pseudocode** statement to declare the enumerated data type, `Vehicle`, to hold the identity code of each of the company's taxis:

M100, M230, T101, T102, T120, T150

.....

..... [2]

(b) Write **pseudocode** statements to declare the composite data type, `Booking`, to hold data about taxi bookings. The data required includes:

- booking number (any combination of letters and numbers)
- destination
- client name
- client telephone number
- date of departure
- address for pick-up
- the identity code of the taxi used.

Use the most appropriate data type in each case, including the enumerated data type from part **(a)**.

.....

.....

.....

.....

.....

.....

.....

..... [4]

1 Data types can be defined using pseudocode.

The composite record data type, `Departure`, is used to represent flights from Cambridge Airport and is defined in pseudocode as:

```
TYPE Departure
   DECLARE FlightNumber : STRING
   DECLARE Destination : STRING
   DECLARE FlightDate : DATE
   DECLARE Gate : STRING
   DECLARE Airline : STRING
ENDTYPE
```

A variable, `Flight1`, is declared in pseudocode as:

```
DECLARE Flight1 : Departure
```

(a) Write **pseudocode** to store the following details to `Flight1`:

Field	Data
FlightNumber	SB2789
Destination	Dublin
FlightDate	30/07/2025
Gate	N03
Airline	Cambridge Airways

.....

.....

.....

.....

.....

..... [3]

(b) The data type for `Gate` is changed to an enumerated data type, `GateID`.

(i) Write a **pseudocode** statement to declare `GateID` to hold the identity codes for the airport gates:

N01, N02, N03, W01, W02, W03, W04

.....

..... [2]

(ii) Write the new **pseudocode** statem equired to replace the declaration of `Gate` in `Departure`.

[1]

1 A programmer is writing a program to manage a video library. They require a user-defined data type.

(a) Write **pseudocode** statements to declare the composite data type `VideoLibrary` to hold data about each video in the collection. This data includes:

- identity code (any combination of letters and numbers)
- title
- year released
- date purchased
- format (for example DVD, Blu-ray, 4K, MP4)
- running time (minutes)

Use the most appropriate data type in each case.

.....

.....

.....

.....

.....

.....

.....

..... [4]

(b) Identify **one** field in `VideoLibrary` that could be an efficient enumerated data type and give a reason for your choice.

Field .....

.....

Reason .....

.....

[2]

9 A veterinary surgery wants to create a class for individual pets.
Some of the attributes required in the class are listed in the table.

Attribute	Data type	Description
`PetID`	`STRING`	unique ID assigned at registration
`PetType`	`STRING`	type of pet assigned at registration
`OwnerTelephone`	`STRING`	telephone number of owner assigned at registration
`DateRegistered`	`DATE`	date of registration

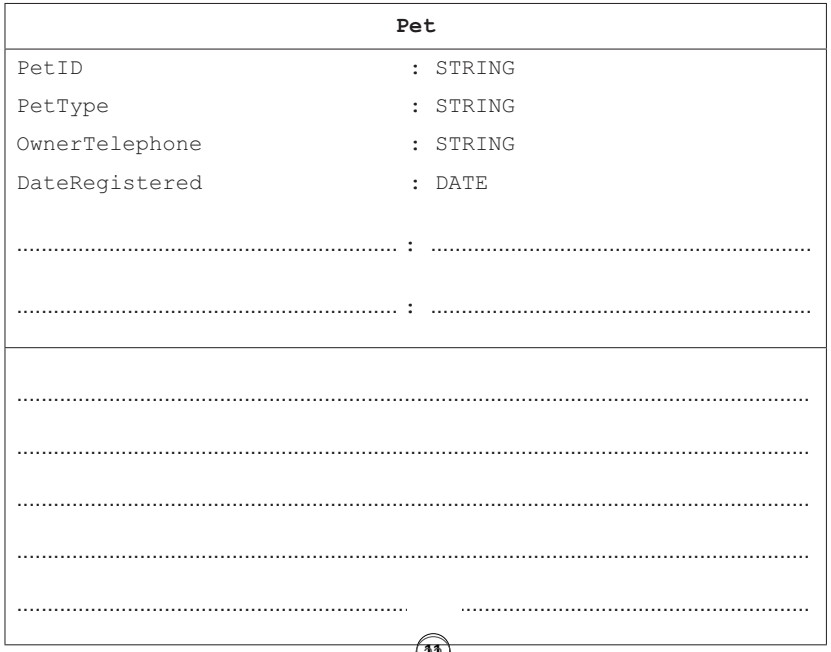
(a) State **one** reason why the attributes would be declared as `PRIVATE`.

.....

..... [1]

(b) Complete the class diagram for `Pet`, to include:

- an attribute and data type for the name of the pet
- an attribute and data type for the name of the owner
- a method to create a `Pet` object and set attributes at the time of registration
- a method to assign a pet ID
- a method to assign the date of registration
- a method to return the pet name
- a method to return the owner's telephone number.



[5]

3 **(a)** Describe the user-defined data type **record**.

.....

.....

.....

.....

.....

..... [3]

(b) A programmer defines a record, `Order`, to store the following data:

- account number
- order number
- order price
- order date.

Write **pseudocode** statements to define this record.

.....

.....

.....

.....

.....

.....

.....

..... [4]

6 **(a)** Describe the user-defined data type **set**.

.....

.....

.....

.....

.....

..... [3]

(b) Write **pseudocode** statements to declare the set data type, `SymbolSet`, to hold the following set of mathematical operators, using the variable `Operators`.

```
+  -  *  /  ^
```

.....

.....

.....

.....

.....

.....

..... [4]

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

13.2 File organisation and access

Name: _____ Class: _____ Date: _____ **Total: 14 marks**

KEY WORDS file organisation 文件组织 • hash function 散列函数 • direct access 直接存取 • sequential access 顺序存取 • record 记录

Objective

Build the skills to answer exam questions on **file organisation and access** — serial, sequential and random files, and **hashing**.

You must be able to:

- describe **serial**, **sequential** and **random** file organisation
- match **sequential** and **direct** access to a problem
- use a **hash function** and resolve a **collision**

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

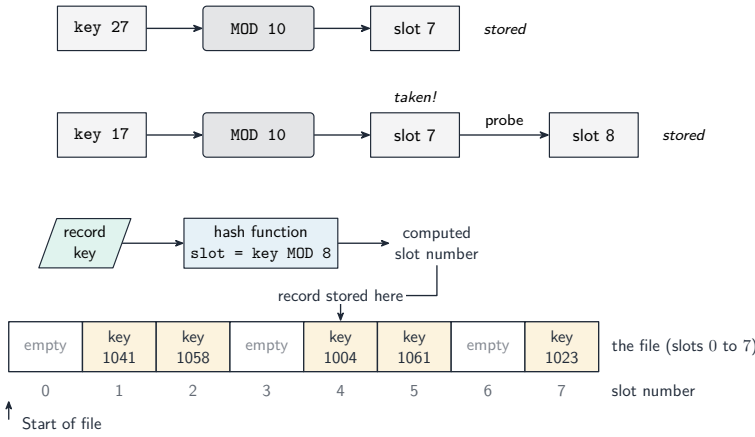
■ File organisation and access

Organisation	Order of records	Best for	Access
serial	order **added**	logs, audit trails	sequential only
sequential	sorted by a **key**	master files, in-order reports	sequential / direct
random	position from the **key** (hash)	fast single-key lookup	direct

Sequential access reads from start to end; **direct access** jumps straight to a known position.

■ Hashing

A **hash function** turns a key into an **address**: e.g. `address ← key MOD N`.



Random (direct) file access by address

A **collision** is when two keys hash to the same slot. Resolve it by **linear probing** (try the next free slot) or **chaining** (each slot holds a linked list). Keep the **load factor** (records ÷ slots) below ~70%.

2 Practice

Now apply the methods above.

2.1 State the difference between a **serial** file and a **sequential** file. [2]

2.2 State which file organisation gives the **fastest single-key lookup**. [1]

2.3 State what a **hash function** does. [1]

2.4 State what a **collision** is. [1]

3 Exam-style questions

3.1 A file uses `address ← key MOD 10`. Give the slot number for the keys `45`, `23` and `30`. [3]

3.2 Keys `45` and `35` both hash to slot 5. Explain how **linear probing** stores the second one. [2]

3.3 State **one** advantage and **one** disadvantage of a **random (direct-access)** file. [2]

3.4 A bank must look up an account **quickly** by its account number. State the best file organisation and **justify** your choice. [2]

Solutions

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

2.1

- a **serial** file keeps records in the **order they were added** (unsorted)
- a **sequential** file keeps them **sorted by a key field**

2.2

- a **random (direct-access)** file

2.3

- it turns a **record key** into an **address/slot** where the record is stored

2.4

- when **two different keys** hash to the **same address/slot**

3.1

- `45 MOD 10 = 5`
- `23 MOD 10 = 3`
- `30 MOD 10 = 0`

3.2

- `35 MOD 10 = 5`, but slot 5 is **taken** by `45`
- linear probing tries the **next slot** (slot 6), and stores `35` there (the first free slot, wrapping if needed)

3.3

- advantage: very **fast direct lookup** by key
- disadvantage: reading in **key order** is harder, and collisions must be handled (space may be wasted)

3.4

- a **random (direct-access)** file using a **hash** of the account number
- because it gives near-instant lookup of a single record without searching the whole file

3 A program stores records in the 2D array `HashTable`. Each record is stored at a specific index of the array that is calculated using a hashing algorithm with the record's key field.

The array has 100 × 10 elements. The hashing algorithm uses the key to generate an index between 0 and 99 (inclusive). If two key fields generate the same index, there is a collision. Any records that have a collision are stored in the next space in the same index.

For example: In this table two record keys generated the same hash value of 1. Four record keys generated the same hash value of 3.

Index	0	1	2	3	4	5	…	9
0	record							
1	record	record						
2								
3	record	record	record	record				
4								
…								
99								

The program uses Object-Oriented Programming (OOP).

(a) The class `Record` stores data about the records:

Record	
`Key : Integer`	stores the integer key field for the data
`Data : String`	stores the string data
`Constructor()`	initialises `Key` and `Data` to its parameter values

The attributes `Key` and `Data` are public.

Write program code to declare the class `Record` and its constructor.

Use your programming language appropriate constructor.

Save your program as **Question3_N25**.

Copy and paste the program code into part **3(a)** in the evidence document.

[2]

(b) The procedure `InitialiseHashTable()` initialises each element in the array to an empty or null record.

Write program code to declare the global 2D array `HashTable` and the procedure `InitialiseHashTable()`

Save your program.

Copy and paste the program code into part **3(b)** in the evidence document.

[2]

(c) The function `Hash()`:

- takes an integer key field as a parameter
- calculates and returns the hash value of the key field.

The hash value is the result from the formula: `key MOD 100`

Write program code for `Hash()`

Save your program.

Copy and paste the program code into part **3(c)** in the evidence document.

[2]

(d) The procedure `InsertData()`:

- takes an object of type `Record` as a parameter
- calculates the hash value for the parameter using the appropriate function
- stores the parameter in the correct position in `HashTable`

You can assume there will be no more than 10 objects that generate the same hash value.

Write program code for `InsertData()`

Save your program.

Copy and paste the program code into part **3(d)** in the evidence document.

[4]

(e) The file "HashTableData.txt" stores 200 key values and string data items in the format:

```
key,string
```

For example, the first row in the text file is:

```
528,permission
```

The key is 528 and the string data is "permission"

The procedure ReadData():

- opens the text file and reads each line
- splits each line into the key and data
- calls InsertData() with an object containing each key and matching data.

Write program code for ReadData()

Save your program.

Copy and paste the program code into part **3(e)** in the evidence document.

[5]

(f) The function GetRecord():

- takes an integer key field as a parameter
- calculates the hash value for the key field using the appropriate function
- searches the hash table for the record with the matching key field
- returns the data for the record if the record is found
- returns "Not found" if the record is not found.

Write program code for GetRecord()

Save your program.

Copy and paste the program code into part **3(f)** in the evidence document.

[5]

(g) The main program:

- calls InitialiseHashTable() and ReadData()
- takes **five** integer key fields as input from the user
- calls GetRecord() with each input and outputs the return value.

(i) Write program code for the main program.

Save your program.

Copy and paste the program code into part **3(g)(i)** in the evidence document.

[3]

(ii) Test your program with the following **five** inputs in the order given:

528

1128

1828

1062

39

Take a screenshot of the output(s).

Save your program.

Copy and paste the screenshot(s) into part **3(g)(ii)** in the evidence document.

[2]

11 The pseudocode algorithm below allows a user to input a new stock item. The random file is searched for the next empty location in the file and the new item is inserted there.
A suitable message is displayed if the file is full.

Complete this pseudocode.

```
DECLARE Location : INTEGER
DECLARE NewStock : STRING
DECLARE CurrentStock : STRING
DECLARE Stored : BOOLEAN
DECLARE Max : INTEGER
Max ← 100000
Stored ← FALSE
Location ← 1

.....
OUTPUT "Enter the new item you wish to store: "
INPUT NewStock
WHILE NOT Stored AND Location <= Max

   .....

   GETRECORD "StockList.dat", .....
   IF CurrentStock = "" THEN

      ..... "StockList.dat", NewStock
      Stored ← TRUE
   ELSE
      Location ← Location + 1
   ENDIF
ENDWHILE

..... THEN
   OUTPUT "The new item has not been stored as the file was full."
ENDIF
CLOSEFILE "StockList.dat"
```

[5]

2 **(a)** Describe **serial file organisation** as a method of storing data records in a file.

.....

.....

.....

..... [2]

(b) State **one** example of a use for serial file organisation.

.....

..... [1]

5 **(a)** Explain what is meant by a hashing algorithm in the context of file access.

.....

.....

.....

.....

.....

..... [3]

(b) The use of a hashing algorithm can result in the same storage location being identified for more than one record.

Outline **two** methods of overcoming this issue.

1 .....

.....

2 .....

.....
[2]

A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

13.3 Floating-point numbers, representation and manipulation

Name: _____ Class: _____ Date: _____ **Total: 21 marks**

KEY WORDS exponent 指数 • mantissa 尾数 • floating-point 浮点 • two's complement 补码 • floating-point numbers 浮点数

Objective

Build the skills to answer exam questions on **floating-point numbers** 浮点数 — the format, conversions, normalisation and rounding error.

You must be able to:

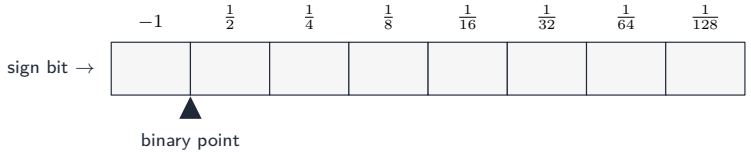
- describe the **mantissa** + **exponent** format
- convert **binary** ↔ **denary** floating-point
- **normalise** a number and explain **rounding errors**

1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

The format

A floating-point number is mantissa $\times 2^{\text{exponent}}$. Both are stored in **two's complement**. The mantissa is a fraction with the binary point just after the **sign bit**:



Binary → denary

Mantissa `01010000`, exponent `00000010`:

- sign bit `0` → positive; fraction `.1010000` $= \frac{1}{2} + \frac{1}{8} = 0.625$
- exponent `00000010` $= +2$
- value $= 0.625 \times 2^2 = 0.625 \times 4 =$ **2.5**

Denary → binary (normalised)

Convert `6.0` (8-bit mantissa, 8-bit exponent):

- `6` in binary $= 110.0$
- move the point left until it is just after the first `1`: 0.110×2^3
- mantissa = `01100000`, exponent $= 3 =$ `00000011`

Normalising

A positive mantissa is **normalised** when it begins `0.1…` (no wasted leading zeros — maximum precision). Shift left and decrease the exponent:

- `00011000` exp `00000101` $= 0.0011 \times 2^5$
- shift mantissa left 2 → `01100000`, exponent $5 - 2 = 3 =$ `00000011` (same value).

Negative numbers

A **negative** mantissa is in **two's complement** and is normalised when it begins `1.0…` (the first two bits differ). To represent -0.75: take $+0.75 =$ `0.1100000`, then two's-complement it (invert, add 1) → `1.0100000`. The sign/integer bit has place value -1, so `1.0100000` $= -1 + \frac{1}{4} = -0.75$.

Rounding error

Some denary reals (like 0.1) are **repeating** binary fractions, so they are stored only **approximately**. Errors build up, so test `ABS(x - 0.3) < 1e-9`, not `x = 0.3`.

2 Practice

Now apply the methods above.

2.1 State the **two** parts of a floating-point number and what each represents. [2]

2.2 A mantissa `01000000` has exponent `00000011`. State its **denary** value. [2]

2.3 State what it means for a number to be **normalised**. [1]

2.4 State **one** reason some denary real numbers cannot be stored exactly. [1]

3 Exam-style questions

A system uses an **8-bit mantissa** and an **8-bit exponent**, both in two's complement.

3.1 Convert mantissa `01010000`, exponent `00000010` to **denary**. Show your working. [3]

3.2 Convert `6.0` to its **normalised** floating-point representation. Show your working. [4]

3.3 Normalise the mantissa `00011000` with exponent `00000101`. Give the new mantissa and exponent. [3]

3.4 Explain why `0.1 + 0.2` might **not** equal exactly `0.3` when stored in floating-point. [2]

3.5 Convert the floating-point number with mantissa `10110000` and exponent `00000010` to **denary**. Show your working. [3]

Solutions

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

2.1

- the **mantissa** holds the **significant digits**
- the **exponent** is the **power of 2** to multiply by

2.2

- `0.1000000` = 0.5; exponent = 3
- $0.5 \times 2^3 = \mathbf{4}$

2.3

- the first significant bit is **immediately after the binary point** (`0.1…` for a positive number) — no wasted leading zeros, so precision is maximised

2.4

- some denary fractions (e.g. 0.1) are **repeating/infinite** in binary, so they must be **truncated** to fit the mantissa

3.1

- sign bit `0` → positive
- `.1010000` = $\frac{1}{2} + \frac{1}{8} = 0.625$, exponent = 2
- $0.625 \times 2^2 = \mathbf{2.5}$

3.2

- `6` = 110.0_2
- normalise to 0.110×2^3
- mantissa `01100000`
- exponent `00000011`

3.3

- `00011000` = 0.0011×2^5; shift mantissa **left 2**, exponent $5 - 2 = 3$
- new mantissa `01100000`
- new exponent `00000011`

3.4

- `0.1` and `0.2` cannot be stored **exactly** in binary (they are repeating fractions)
- the small stored errors **add up**, so the result is very close to but **not exactly** `0.3`

3.5

- sign bit `1` → negative
- `1.0110000` = $-1 + \frac{1}{4} + \frac{1}{8} = -0.625$, exponent = 2
- $-0.625 \times 2^2 = -\mathbf{2.5}$

2 Numbers are stored in a computer system using binary floating-point representation with:

- 12 bits for the mantissa
- 4 bits for the exponent
- two's complement form for both the mantissa and the exponent.

(a) Write the normalised floating-point representation of the following positive binary number using this system.

0.00000001110101101

Mantissa **Exponent**

[2]

(b) Calculate the normalised binary floating-point representation of –76.1875 in this system. Show your working.

Mantissa **Exponent**

Working .....

.....

.....

.....

.....

.....

.....

.....

[4]

2 Numbers are stored in a computer using binary floating-point representation with:

- 10 bits for the mantissa
- 6 bits for the exponent
- two's complement form for both the mantissa and the exponent.

(a) Write the normalised floating-point representation of the following binary number using this system.

0.00000011010111

Mantissa **Exponent**

[2]

(b) Calculate the normalised binary floating-point representation of –25.3125 in this system. Show your working.

Mantissa **Exponent**

Working .....

.....

.....

.....

.....

.....

.....

.....

[4]

2 Numbers are stored in a computer using binary floating-point representation with:

- 12 bits for the mantissa
- 4 bits for the exponent
- two's complement form for both the mantissa and the exponent.

(a) Calculate the normalised binary floating-point representation of +124.4375 in this system.
Show your working.

Mantissa												Exponent			

Working .....

.....

.....

.....

.....

.....

[3]

(b) Calculate the denary value of the following normalised binary floating-point number.
Show your working.

Mantissa												Exponent			
1	0	1	0	0	0	1	0	1	0	1	1	0	1	1	0

Working .....

.....

.....

.....

.....

Denary value .....

[3]

1 Numbers are stored in a computer using binary floating-point representation with:

- 10 bits for the mantissa
- 6 bits for the exponent
- two's complement form for both the mantissa and the exponent.

(a) Calculate the normalised binary floating-point representation of +201.125 in this system.

Show your working.

Mantissa										Exponent					

Working .....

.....

.....

.....

.....

.....

[3]

(b) Calculate the denary value of the given normalised binary floating-point number.

Show your working.

Mantissa										Exponent					
1	0	1	0	1	1	0	0	1	1	0	0	0	1	0	1

Working .....

.....

.....

.....

.....

.....

Answer .....

[3]

4 Numbers are stored in a computer using binary floating-point representation with:

- 12 bits for the mantissa
- 4 bits for the exponent
- two's complement form for both the mantissa and the exponent.

(a) Calculate the denary value of the given normalised binary floating-point number.

Show your working.

Mantissa												Exponent			
0	1	0	0	0	1	1	1	0	1	1	1	0	1	1	1

Working .....

.....

.....

Answer .....

[2]

(b) Calculate the normalised binary floating-point representation of –49.1875 in this system.

Show your working.

Mantissa												Exponent			

Working .....

.....

.....

.....

.....

.....

.....

.....

[4]