1

**a)** [18]

**Ans:**

|  |
| --- |
| Employee |
| Private:  EmpID  Name  PhoneNo  Type |
| Public:  getEmpID()  getName()  getPhoneNo()  getType()  setEmpID()  setName()  setPhoneNo()  setType()  Payroll()  Display() |

|  |
| --- |
| Hourly\_paid\_employee |
| Private:  HourlyRate  HoursWorked |
| Public:  getHourlyRate ()  getHoursWorked ()  setHourlyRate()  setHoursWorked ()  Payroll()  Display() |

|  |
| --- |
| Salaried\_employee |
| Private:  Salary |
| Public:  getSalary ()  setSalary ()  Payroll()  Display() |

**(b) Ans: Batch count** [1]

**(c)** **Ans:** Length check: 8, format check [every character is a digit] [2]

**(d) Ans: object diagram**  [2]

**CS (e)** **Ans:** This is known as polymorphism.

Polymorphism usually means that classes Hourly\_paid\_employee and Salaried\_employee derive from class Employee, or class Hourly\_paid\_employee and Salaried\_employee implements an interface that represents class Employee.

The primary usage of polymorphism is the creation of objects belonging to different classes to respond to method [Payroll()], field, or property calls of the same name, each one according to an appropriate class-specific The programmer (and the program) does not have to know the exact type of the object in advance, and so the exact behaviour is determined at run time.

**(f)** Any two benefits and shortcomings each for using native and web applications. [8]

**Ans: Benefits of Native Applications**

**Optimum Access to Device Hardware and Features**

**Potential for High Performance and High Responsiveness**

**Enhanced User Experience**

**Offline Capabilities**

**Ans: Shortcomings of Native Applications**

**Longer Development Time and Higher Development Costs**

**App Store Approval**

**Limited Distribution and Discoverability Outside App Stores**

**Fragmentation and Compatibility Challenges**

**Ans: Benefits of Web Applications**

**Cross-Platform Compatibility**

**No Requirements for Installation**

**Updating and Application Maintenance**

**Cost-Effective Development**

**Greater Discoverability**

**Ans: Shortcomings of Web Applications**

**Internet Connectivity Dependency**

**Limited Device Feature Access**

**Performance Constraints**

**Security considerations**

**(g)** [4]

**Ans:**

**1. Visibility of System Status**

**2. Match between system and the real world**

**3. User control and freedom**

**4. Consistency and standards**

**5. Error prevention**

**6. Recognition rather than recall  
7. Flexibility and efficiency of use**

**8. Aesthetic and minimalist design**

**9. Help users recognize, diagnose, and recover from errors**

**10. Help and documentation**

*Even though it is better if the system can be used without documentation, it may be necessary to provide help and*

**(h)** **Ans:** [3]

* safer since less chance of external hacking or viruses
* can prevent workers accessing unwanted sites?
* can ensure information is specific to the company
* easier to send out “sensitive” messages to remain within company only

**2** **(a**) **Ans**

When searching an ordered list the search can be terminated when an item greater than the search value (or less than) is reached **[1]**

When searching an unordered list the search cannot be terminated until the last item has been reached.

For an ordered list a binary search can be used. **[1]**

**(b)** The table below includes an unordered list of maximum 10 names.

**Ans:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **Name** | **Next Pointer (1)** | **Next Pointer (2)** | **Next Pointer (3)** |
| 0 | Smith | 4 |  |  |
| 1 | Jones | 3 |  |  |
| 2 | Ahmed | 5 |  |  |
| 3 | Lewis | 0 |  |  |
| 4 | Thomas | **null** |  |  |
| 5 | Brown | 1 |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |

**(c)** [4]

**Ans:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **Name** | **Next Pointer (1)** | **Next Pointer (2)** | **Next Pointer (3)** |
| 0 | Smith |  | 4 |  |
| 1 | Jones |  | 3 |  |
| 2 | Ahmed |  | 5 |  |
| 3 | Lewis |  | 6 |  |
| 4 | Thomas |  | **null** |  |
| 5 | Brown |  | 7 |  |
| 6 | Murphy |  | 0 |  |
| 7 | Collins |  | 1 |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |

**(d)** [2]

**Ans:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Index** | **Name** | **Next Pointer (1)** | **Next Pointer (2)** | **Next Pointer (3)** |
| 0 | Smith |  |  | 4 |
| 1 | Jones |  |  | 3 |
| 2 | Ahmed |  |  | 5 |
| 3 | Lewis |  |  | 6 |
| 4 | Thomas |  |  | **null** |
| 5 | Brown |  |  | 7 |
| 6 | Murphy |  |  | 4 |
| 7 | Collins |  |  | 1 |
| 8 |  |  |  |  |
| 9 |  |  |  |  |

**(e)** **Ans:** [3]

* Traverse the list until the last Name was found. Then insert the new Name to the end of the list. Name that have to consider special cases such as list being empty or full.
* In case of a list being empty, returning the updated head of the linked list because in this case, the inserted Name is the first as well as the last Name of the linked list.
* In case of the list is being full, returning with message "List is full. Insert failed!"

**(f)** **Ans:** [3]

This can be achieved by linking up the unused nodes to form another linked list, which becomes the **free space list**. A free space list needs to be used for maintaining the unused elements in the table.



**(g)** [3]

**3 (a) (i)** [1]

|  |
| --- |
| **Ans: Recipe table; A Figure 2; [1]** |

**(ii)** [1]

|  |
| --- |
| **Ans: contains multiple values in Ingredients field/attribute/column [1]**  **// data in Ingredients column not atomic // repeating groups;** |

**(b)** **(i)** [1]

|  |
| --- |
| ***fully normalised:***  **Ans: every attribute is dependent on the key, the whole key and nothing but the key; [1]** |
| **OR (tables contain no repeating groups of attributes,) no partial dependencies;**  **no non-key dependencies;** |

**(ii)** [1]

|  |
| --- |
| **Ans: to aid consistency of data // to avoid potential data inconsistency problems [1]** |
| **// to eliminate data inconsistency // to minimise data duplication** |

**(c)** **(i)** Recipe(

|  |
| --- |
| **Ans:  *RecipeID, Dish, PrepTime, CookTime, NoOfServings, CookInstructions* [1]** |

*)*

[1]

**(ii)** FoodItem( ........................................................................................................

|  |
| --- |
| **Ans: *FoodItemID, FoodItemName, PackSize, Price* [1]** |

..)

[1]

**(iii)** RecipeIngredient(

|  |
| --- |
| **Ans:  *FoodItemID, RecipeID, Quantity* [2]** |

..)

**(d) Ans: SELECT FoodItemName, Quantity, PackSize, Price**

**FROM FoodItem, RecipeIngredient, Recipe**

**WHERE (Recipe.RecipeId = RecipeIngredient.RecipeId)**

**AND (RecipeIngredient.FoodItemId = FoodItem.FoodItemId)**

**AND (Recipe.Dish = “Feta Salad”)**

**ORDER BY FoodItemName ASC**

**(e) Ans:**  [3]

* Access rights give chef/assistants access to different elements

… by having different accounts / logins

… which have different access rights e.g. read only // no access / read /write

* Specific views can be assigned to chef/assistants

… e.g. assistants can only see the data for their own area(s)

**4(a)** **Ans: One** mark for each correct marking point [3]

* The initial order of the data
* The number of data items to be sorted
* The efficiency of the sorting algorithm

**(b)** **Ans: One** mark for each marking point

1 Use of FOR loop to cycle through the whole year group

2 Temporary storage of the score being ‘inserted’

3 Temporary storage of the corresponding name elements

4 Use of WHILE loop with correct exit clause

5 Moving of all three elements of data to next array elements

6 Correct updating of counter variable

7 Final insertion of all three data elements

**Example algorithm**

YearSize ← 249

FOR Student ← 2 to YearSize

Temp1 ← Score[Student]

Temp2 ← Name[Student]

Counter ← Student

WHILE Counter > 1 AND Score[Counter - 1] < Temp1

Score[Counter] ← Score[Counter - 1]

Name[Counter] ← Name[Counter - 1]

Counter ← Counter – 1

ENDWHILE

Score[Counter] ← Temp1

Name[Counter] ← Temp2

NEXT Student

**(c)**

|  |
| --- |
| **((i) Ans: O(N²)** [1] |

**(ii)** [2]

|  |
| --- |
| **Ans: O(N²)Both the same** |

**5**  **(a)** **Ans:** [2]

sender’s IP address

receiver’s IP address

packet sequence number

checksum

**(b)** **Ans:** [3]

email has been split up into packets

packet has destination IP address

packets pass through many different routers in journey

packets don’t take same route

routers use IP addresses

packets reassembled at destination to rebuild email

**(c)** **Ans:** [2]

email message is only read when all of it is received

time delays due to lost / delayed packets not significant

so sending different packets by different routes is not issue / is efficient

packets arriving out of order not an issue

no requirement for a continuous circuit (circuit switching)

**(d) Ans:** Circuit switching [1]

**(e) Ans:** (eg real-time video / video conferencing [2]

circuit made available is dedicated to this communication stream

full bandwidth available / no sharing

no lost packets

guaranteed quality of service