2024/NJC/1

2024/NJC/1/1 [8]
2024/NJC/1/2 [20]
2024/NJC/1/3 [12]
2024/NJC/1/4 [12]
2024/NJC/1/5 [24]
2024/NJC/1/6 [17]
2024/NJC/1/7 [7]

2024/NJC/1/1 [8]

▼ 1(a) [2]

- [1] A queue is a First In First Out (FIFO) data structure. Items are inserted at the end/back of the queue (Enqueue operation),
- [1] while the first item (head of the queue) in the queue will be removed first (Dequeue operation).

▼ 1(b) [2]

Choose 2 out of 4 in the table

- [1] For 1 correct
- [1] For another correct

	Linear Queue	Circular Queue
1	Arranges the data in a linear pattern.	Arranges the data in a circular order where the rear end is connected with the front end.
2	The insertion and deletion operations are fixed i.e, done at the rear and front end respectively.	Insertion and deletion are not fixed and it can be done in any position.
3	In the case of a linear queue, the element added in the first position is going to be deleted in the first position. The order of operations performed on any element is fixed i.e., FIFO.	In the case of circular queue, the order of operations performed on an element may change.
4	Front elements cannot be reused once a dequeue operation takes place.	Space in front can be reused when the tail wrap around to the front.

▼ 1(c) [4]

- [1] Check if queue is full
- [1] If queue is empty, initialize head and tail
- [1] Move tail to next position in circular manner
- [1] Insert value at the tail of the queue

```
Function Q_Q(queue, value, head, tail)

    // [1] Check if queue is full
    If ((tail + 1) % 100 == head)
        Return False  // Queue is full
    // [1] If queue is empty, initialize head and tail
    If (head == -1)
        head = 0
        tail = 0
    Else
        // [1] Move tail to next position in circular manner
        tail = (tail + 1) % 100

    // [1] Insert value at the tail of the queue
    queue[tail] = value

    Return True
End Function
```

```
def Q_Q(queue, value, head, tail):

    # [1] Check if queue is full
    if ((tail + 1) % 3 == head):
        return False

    # [1] If queue is empty, initialize head and tail
    if (head == -1):
        head = 0
        tail = 0
    else:
        # [1] Move tail to next position in circular manner
        tail = (tail + 1) % 3

    # [1] Insert value at the tail of the queue
    queue[tail] = value
    print(head,tail)

    return (True,(head,tail))

#Tester code
queue = [None, None, None]

head = -1
tail = -1

print(head, tail)
Q_Q(queue,0,head,tail)
print(queue)
head, tail = Q_Q(queue,0,head,tail)[1]
print(head,tail)
Q_Q(queue,1,head,tail)
print(queue)
head, tail = Q_Q(queue,1,head,tail)[1]
```

```
print(head,tail)
Q_Q(queue,2,head,tail)
print(queue)
head, tail = Q_Q(queue,2,head,tail)[1]
print(head,tail)
print(Q_Q(queue,3,head,tail))
print(queue)
```

2024/NJC/1/2 [20]

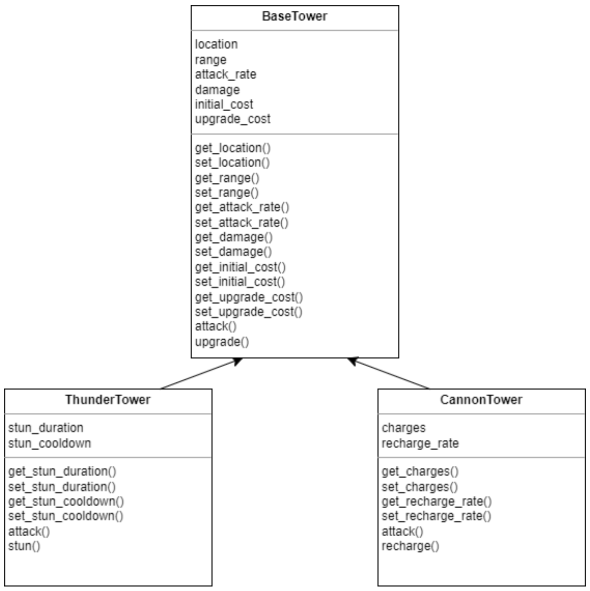
▼ 2(a) [2]

- [1] A class is a template for an object, and it defines the attributes and methods of objects in that class.
- [1] An object is an instance of a class.

▼ 2(b) [12]

- [1] 3 classes observed `BaseTower` , `ThunderTower` and `CannonTower` put in 3 rectangles
- [1] `ThunderTower` and `CannonTower` has arrows pointing to `BaseTower` to show their status as child class
- [1] all attributes of `BaseTower` given
- [1] at least 3 pairs of getter and setter methods for attributes of `BaseTower`
- [1] all 6 pairs of getter and setter methods for all attributes of `BaseTower`
- [1] `attack()` and `upgrade()` method of `BaseTower`
- [1] extra attributes `stun_duration` and `stun_duration` of `ThunderTower` given
- [1] all getters and setters method for extra attributes of `ThunderTower`
- [1] `attack()` and `stun()` method of `ThunderTower`
- [1] extra attributes `charges` and `recharge_rate` of `CannonTower` given
- [1] all getters and setters method for extra attributes of `CannonTower`
- [1] `attack()` and `recharge()` method for `CannonTower`

uml.drawio



▼ 2(c) [2]

- [1] range check
- [1] null check/type check

▼ 2(d) [2]

- [1] Inheritance is the ability of a subclass to derive properties and methods from its superclass.

- [1] In this example, the `location` , `range` , `damage` attributes of of the parent class `BaseTower` is also present in the child classes `ThunderTower` and `CannonTower` without needing to redefine them

▼ 2(e) [2]

- [1] Polymorphism is a concept in object-oriented programming (OOP) where different objects can respond to the same method call in different ways, based on their type.
- [1] In this example, the same method `attack()` behave differently depending on the type the towers. `ThunderTower` objects deals no damage unlike the `BaseTower` objects

2024/NJC/1/3 [12]

▼ **3(a) [6]**

- [1] If the tree is empty, return -1
- [1] Initialize current pointer to root
- [1] Traverse the tree
- [1] If the current node's data matches the value, return the pointer (index)
- [1] If the value is smaller, move to the left subtree or If the value is larger, move to the right subtree
- [1] If the value is not found, return -1

```
Function find(tree, value, root)

    // [1] If the tree is empty, return -1
    If root == -1
        Return -1

    // [1] Initialize current pointer to root
    current = root

    // [1] Traverse the tree
    While current != -1
        // [1] If the current node's data matches the value, return the pointer (index
        If tree[current][1] == value
            Return current

        // [1] If the value is smaller, move to the left subtree
        Else If value < tree[current][1]
            current = tree[current][0]

        // If the value is larger, move to the right subtree
        Else
            current = tree[current][2]

    // [1] If the value is not found, return -1
    Return -1
End Function
```

```
def find(tree, value, root):

    #  [1] If the tree is empty, return -1
    if root == -1:
        return -1

    # [1] Initialize current pointer to root
    current = root

    # [1] Traverse the tree
    while current != -1:
        # [1] If the current node's data matches the value, return the pointer (inde
x)
        if tree[current][1] == value:
            return current

        # [1] If the value is smaller, move to the left subtree
        elif value < tree[current][1]:
            current = tree[current][0]

        # If the value is larger, move to the right subtree
        else:
            current = tree[current][2]

    # [1] If the value is not found, return -1
    return -1
```

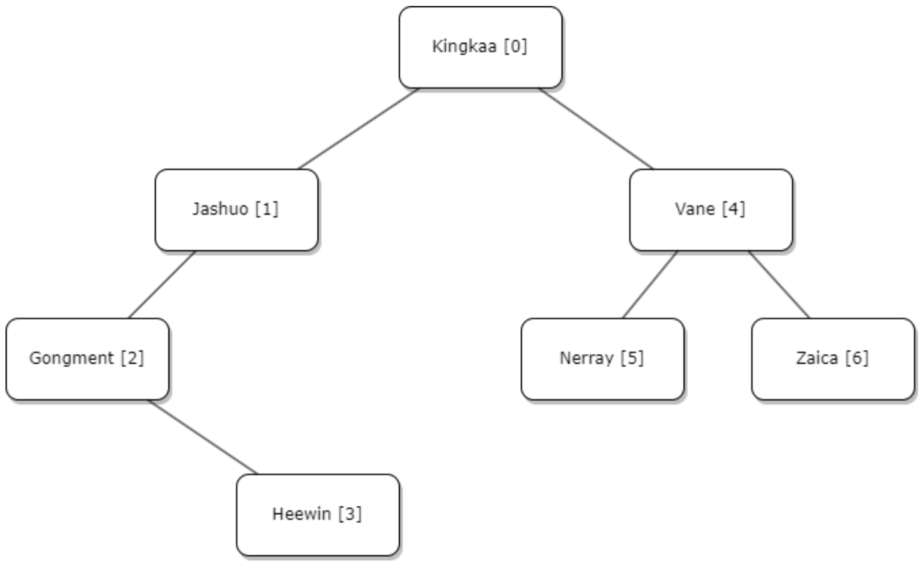
▼ **3(b) [3]**

- [1] at least 2 entry correct
- [1] at least 4 entry correct
- [1] all 6 entry correct

Index	left	data	right
0	1	Kingkaa	4
1	2	Jashuo	-1
2	-1	Gongment	3
3	-1	Heewin	-1
4	5	Vane	6
5	-1	Nerray	-1
6	-1	Zaica	-1

▼ **3(c) [2]**

- [1] Correct Left Subtree
- [1] Correct Right Subtree



▼ **3(d) [1]**

- [1] Same as original order: Kingkaa, Jashuo, Gongment, Heewin, Vane, Neeray, Zaica

2024/NJC/1/4 [12]

▼ 4(a) [2]

- [1] Equipment B is the switch as ...
- [1] ... a switch connects devices within a network, (alternatively) while a router connects different networks to each other.

▼ 4(b) [1]

- [1] The internet is a packet switched network

▼ 4(c) [3]

- **[1] Segmentation of Data:** Data is divided into packets. Each packet contains a portion of the data and the destination address.
- **[1] Transmission of Packets:** These packets are transmitted independently and can travel through different paths in the network, unlike the single path used in circuit switching. The chosen path for each packet can change based on current network conditions, such as congestion or link failures.
- **[1] Packets Reassembling:** Packets, therefore, may arrive out of sequence. The recipient takes each packet, makes a note of its number and assembles them into the correct order.

▼ 4(d) [2]

- [1] In a DoS attack, the attacker sends an overwhelming number of requests to the target server.
- [1] As the server can only handle a finite amount of traffic at any given time. By flooding it with more traffic than it can manage, the attacker exhausts the server's resources (e.g., bandwidth, memory, processing power), causing it to slow down, crash, or become unavailable to users.

▼ 4(e) [2]

- **Either, Traffic Filtering and Rate Limiting**:

    - [1] A firewall can be configured to monitor incoming traffic and identify unusual patterns, such as an excessive number of requests from a single IP address or a specific protocol (e.g., ICMP or SYN requests).
    - [1] It can block or throttle this traffic, preventing the server from being overwhelmed by a flood of requests. By limiting the rate of incoming traffic or connections, the firewall helps preserve server resources and maintains normal operation during potential attacks.
- **Or, IP Blacklisting and Whitelisting**:
    - [1] Firewalls can block traffic from known malicious IP addresses or geographical regions associated with attacks by maintaining a blacklist.
    - [1]They can also be set up to allow only trusted IP addresses to access the file server (whitelisting), reducing the risk of unauthorized or malicious traffic overwhelming the server.

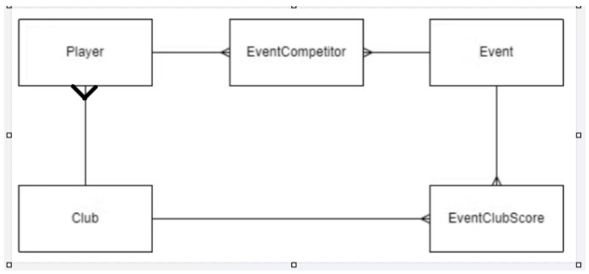
▼ 4(f) [2]

- [1] The student can do an off-site backup of her files, where..
- [1] additional copies of her data are made. These copies can be used to recover primary version of the data if they're or corrupted.

2024/NJC/1/5 [24]

▼ 5(a) [4]

- [1] Correct one to many relationship between `Player` and `Club`
- [1] Correct one to many relationship between `Player` and `EventCompetitor`
- [1] Correct one to many relationship between `Event` and `EventCompetitor`
- [1] Correct one to many relationships between `EventClubScore` to both `Club` and `Event`



▼ 5(b) [10]

- [1] Correct Attributes for `Player` Table
- [1] Correct primary and foreign key for `Player` Table
- [1] Correct Attributes for `Club` Table
- [1] Correct primary and foreign key for `Club` Table
- [1] Correct Attributes for `Event` Table
- [1] Correct primary and foreign key for `Event` Table
- [1] Correct Attributes for `EventCompetitor` Table
- [1] Correct primary and foreign key for `EventCompetitor` Table
- [1] Correct Attributes for `EventClubScore` Table
- [1] Correct primary and foreign key for `EventClubScore` Table

```
Player(PlayerID, PlayerName, ClubName*)
Club(ClubName, SecretaryName, SecretaryEmail)
Event(EventNum, Date, StartTime, Location)
EventCompetitor(CompetitorNum, EventNum*, ClubName*, PlayerName,  MatchScore)
EventClubScore(ClubName*, EventNum*, ClubScore)
```

▼ 5(c) [6]

amat_tennis.db

- [1] using `SUM(Score)` in the `SELECT` statement
- [1] `EventCompetitor.ClubName, SecretaryEmail, SecretaryName` in the `SELECT` statement
- [1] using `INNER JOIN` on `Club.ClubName = EventCompetitor.ClubName`
- [1] use `WHERE EventNumber = 1`
- [1] use `GROUP BY EventCompetitor.ClubName`
- [1] use `ORDER BY SUM(Score) DESC`

```
SELECT SUM(Score), EventCompetitor.ClubName, SecretaryEmail, SecretaryName FROM EventC
INNER JOIN Club ON Club.ClubName = EventCompetitor.ClubName
WHERE EventNumber = 1
GROUP BY EventCompetitor.ClubName
ORDER BY SUM(Score) DESC
```

```
SELECT Club.ClubName, EventClubScore.ClubScore, Club.SecretaryName, Club.SecretaryEm
ail
FROM Club
INNER JOIN EventClubScore
ON EventClubScore.ClubName = Club.ClubName
WHERE EventClubScore.EventNumber = 1
ORDER BY EventClubScore.ClubScore
DESC
```

```
SELECT EventClubScore.ClubName, EventClubScore.ClubScore, Club.SecretaryName, Club.S
ecretaryEmail
FROM Club, EventClubScore
WHERE EventClubScore.EventNumber = 1 AND EventClubScore.ClubName = Club.ClubName
ORDER BY EventClubScore.ClubScore DESC
```

▼ 5(d) [2]

- [1] Minimizing **Data Redundancy.** Normalization ensures that duplicate data is minimized or eliminated across tables. By organizing data into related tables, it reduces the chances of storing the same information multiple times, which saves storage space and improves data consistency.

- [1] **Ensuring Data Integrity.** Normalization helps maintain the accuracy and reliability of the data by organizing tables to minimize update anomalies. This means that when data is inserted, updated, or deleted, the changes are reflected accurately across the database without causing inconsistency or loss of data integrity.

▼ 5(e) [2]

- **[1] It must be in Second Normal Form (2NF).** This means that the table must not have any partial dependency, meaning that all non-key attributes must depend on the whole primary key, not just part of it (in case of a composite key).
- **[1] There must be no transitive dependencies**: A non-key attribute must not depend on another non-key attribute. In other words, every non-key attribute should be directly dependent on the primary key and not on any other non-key attributes.

2024/NJC/1/6 [17]

▼ 6(a) [2]

```
Function z(low, high, seek)
    If low > high Then
        Return -1  // Base case: Element not found

    mid ← (low + high) // 2  // Calculate the middle index

    If A[mid] = seek Then
        Return mid  // Base case: Element found

    Else If A[mid] > seek Then
        Return z(low, mid - 1, seek)  // Recursive case: Search left half

    Else
        Return z(mid + 1, high, seek)  // Recursive case: Search right half
```

- [1] X: `(low + high) // 2`
- [1] Y: `A[mid] > seek`

▼ 6(b) [3]

▼ 6b (i)

- [1] A **recursive function** is a function that calls itself during its execution.

▼ 6(b) (ii)

- [1] Line 10 **and** 12

▼ 6(b) (iii)

- [1] They are the stopping conditions or base case of the recursive function

▼ 6(c) [4]

- [1] Each time a recursive function is called, a new stack frame, containing function's local variables, parameters, and the return address (the point to return to after the function finishes execution), is created.

- [1] The newly created stack frame is pushed onto the call stack. This means that the execution context of the current function call is saved at the top of the stack. The stack grows with each recursive call.
- [1] The execution of the current function is suspended. The new function call is executed with its own stack frame at the top of the stack.
- [1] When a base case is reached or the function completes its task, the function returns a value. The current stack frame is then popped from the stack, and the execution resumes from the return address saved in the stack frame of the previous function call. This process continues until all recursive calls are resolved and the stack is empty again.

▼ 6(d) [3]

- [1] **Base Case(s):** A condition that stops the recursion by providing a straightforward solution without making further recursive calls. This prevents infinite recursion and allows the function to eventually return a result.
- **[1] Recursive Case(s):** A part of the function where it calls itself with modified arguments, gradually moving towards the base case. This ensures the problem is broken down into smaller, more manageable sub-problems.
- **[1] Convergence (Progress Towards the Base Case):** Each recursive call should modify the arguments in such a way that they move closer to the base case. This guarantees that the recursion will terminate after a finite number of steps.

▼ 6(e) [2]

- [1] At least 2 rows are correct with -1 as an output in of the 4 rows
- [1] All entries are correct

Function Call	low	high	seek	mid	A[mid]	OUTPUT
1	0	7	96	3	144	
2	0	2	96	1	-3	
3	2	2	96	2	500	
4	2	1	96			-1
2	0	2	96	1	-3	-1
1	0	7	96	3	144	-1

▼ 6(f) [1]

- [1] The array is not sorted which is a requirement for a binary search algorithm to be used on.

▼ 6(g) [2]

- [1] Recursive functions, especially with large input sizes like a large array, can lead to deep recursion. If the recursion depth becomes too large, it can result in a **stack overflow error.**
- [1] Iterative approaches generally have **lower extra computational resources (such as time and memory) needed** compared to recursion. This usually translates to more efficient use of memory and possibly faster execution, especially for large datasets.

2024/NJC/1/7 [7]

▼ 7(a) [1]

[1] Any 4 digit integers that differs by 3, e.g., 1002, 1005

▼ 7(b) [2]

[1] Chaining can be used to handle the consequence of a collision. In chaining, records are stored in a node of one of the many singly-linked lists. Each array item of the hash table will store the address of one singly-linked list.

[1] Whenever a key of a record gets hashed to an address where another record has already been hashed to, the new record will be added to the end of the list that its hashed address points to.

▼ 7(c) [2]

[1] **Obtaining Consent for Data Collection**: The company must ensure that it collects, uses, or discloses personal data of their customers, e.g., name, address and credit card number, with the user's informed consent. This can be done by clearly informing users about the purpose of collecting their personal data and obtaining their explicit agreement before collecting it. The company should also provide an option to withdraw consent at any time.
[1]
Implementing Data Security Measures: The company must put in place appropriate security measures to protect personal data they collected, e.g., name, address and credit card number, from unauthorized access, misuse, or breaches. This includes implementing encryption, access controls, regular security audits, and ensuring that only authorized personnel have access to sensitive data. This helps in safeguarding the integrity and confidentiality of personal data in compliance with the PDPA.

▼ 7(d) [2]

The team leader is **not abiding** by the code of ethics of a computer professional for the following reasons:

1. **Failure to Prioritize Public and Client Interest**: A core principle of most professional codes of ethics, such as those set by the ACM or IEEE, is to prioritize the safety and welfare of the public and the interests of clients. Data breaches potentially compromise sensitive customer information, which could lead to significant harm, including identity theft or financial loss. By delaying the response to the breach, the team leader is neglecting their duty to protect the privacy and security of customer data, putting the company's clients at risk.
2. **Neglect of Professional Responsibility**: A computer professional is ethically responsible for maintaining the integrity, security, and confidentiality of systems and data. Ignoring a data breach for the sake of completing another project indicates a failure to act responsibly. The team leader should immediately address the breach to mitigate potential harm, rather than postponing a critical issue that could lead to further damage.