SOLUTION QUIZ 01 PAPER B

MCQS (5 MARKS)

1. C) Allows for flexibility in implementing data structures.
2. B) It requires contiguous memory allocation.
3. C) It points to NULL or None.
4. C) Requires less memory per node.
5. A) A pointer to a memory address of zero.

SHORT Q/A

Question 1

Answer:

- Singly linked lists provide flexibility with memory usage and avoid the need for resizing.
- Singly linked lists don't waste memory on unused or empty elements, unlike arrays that may have unused slots.
- Singly linked lists are more memory efficient because they only use the amount of memory required for the elements they store.

Question 02

Answer

Steps to Delete a Node at Position `n` in a Doubly Linked List:

1. **Check if the List is Empty:**
   - If the list is empty (`head == NULL`), there's nothing to delete, and you can simply return.
2. **Traverse to the nth Node:**
   - Use a temporary pointer to traverse the list until you reach the node at position `n`. If the position `n` is greater than the length of the list, return or handle it as an invalid position.
3. **Delete the Node:**
   - Depending on the position, handle three cases:
     1. **Deleting the head node** (position `0`).
     2. **Deleting a node in the middle or end** (position `n > 0`).
4. **Adjust Pointers:**
   - For the node before and after the node to be deleted, adjust the `prev` and `next` pointers to bypass the node being deleted.
5. **Free the Node's Memory:**

- After adjusting the pointers, free the memory occupied by the node to avoid memory leaks.

CODE (5 MARKS, 25 EACH)

Question 01

```
// Function to delete a node at any position
void deleteNodeAtAny(int position) {
if (head == nullptr) {
cout << "List is empty" << endl;
return;
}

if (position == 0) {
Node* temp = head;
head = head->next;
if (head != nullptr) {
head->prev = nullptr;
}
delete temp;

return;
}

Node* temp = head;
for (int i = 0; i < position && temp != nullptr; ++i) {
```

```
temp = temp->next;
}

if (temp == nullptr) {
cout << "Position out of bounds" << endl;
return;
}

if (temp->prev != nullptr) {
temp->prev->next = temp->next;
}
if (temp->next != nullptr) {
temp->next->prev = temp->prev;
}
delete temp;
}
```

Question 02

Error: The code does not update the `head` pointer when inserting at position `0`. The `head` remains pointing to the old first node instead of the new node.

Correction: Update the `head` when inserting at position `0`.

```
void insertAtPosition(Node*& head, int data, int position) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = NULL;

    if (position == 0) {
        newNode->next = head;  // Insert new node at head
        head = newNode;        // Update head to point to new node
    } else {
        Node* temp = head;
        for (int i = 0; i < position - 1 && temp != NULL; i++) {
            temp = temp->next;
        }

        if (temp == NULL) return;  // Position is out of range

        newNode->next = temp->next;
        temp->next = newNode;
    }
}
```