DDS - PYQ Solution

Q.1 (A) Answer the following in one line.

1. **List down the names of data structures which are linear in nature.**
   - **Answer:** Arrays, Linked Lists, Stacks, and Queues are linear data structures.
2. **Given arrays and linked lists, which one of them uses static memory allocation?**
   - **Answer:** Arrays use static memory allocation.
3. **Given arrays and linked lists, which one of them uses dynamic memory allocation?**
   - **Answer:** Linked lists use dynamic memory allocation.
4. **Given arrays and linked lists, which one allows direct access of their elements?**
   - **Answer:** Arrays allow direct access of their elements.
5. **If postponement is the nature of processing, which data structure should be used?**
   - **Answer:** A Queue should be used if postponement is the nature of processing.

(B) Fill in the blanks, in the following sentences.

1. **A self-referential structure has a _____ to itself as one of its members.**
   - **Answer:** A self-referential structure has a **pointer** to itself as one of its members.
2. **A self-referential structure can _____ two such structures together.**
   - **Answer:** A self-referential structure can **link** two such structures together.
3. **A stack behaves in a _____ manner.**
   - **Answer:** A stack behaves in a **LIFO (Last In, First Out)** manner.
4. **A queue behaves in a _____ manner.**
   - **Answer:** A queue behaves in a **FIFO (First In, First Out)** manner.
5. **A linked list is a _____ data structure.**
   - **Answer:** A linked list is a **dynamic** data structure.

Q.2 Attempt any four (Short Questions)

(12 Marks)

1. **Explain primitive data types in short.**
   - **Answer:** Primitive data types in C are basic types provided by the language that represent single values. They include:
     - `int`: Represents integers (e.g., 0, -1, 42).
     - `float`: Represents floating-point numbers (e.g., 3.14, -0.001).
     - `char`: Represents single characters (e.g., 'a', 'B').
     - `double`: Represents double-precision floating-point numbers.

- `void`: Represents the absence of type.

2. **Explain the concept of linear data structures, in short.**
   - **Answer:** Linear data structures in C are data structures where elements are arranged in a sequence, one after another. They allow traversal of elements in a single level, with each element connected to its adjacent elements. Examples include Arrays, Linked Lists, Stacks, and Queues.
3. **Explain and evaluate the following postfix expression:** `2 3 + 7 5 *`
   - **Answer:**
     - **Step 1:** Evaluate `2 3 +`, which gives `5`.
     - **Step 2:** Evaluate `7 5 *`, which gives `35`.
     - **Final Expression:** The result is `5 35`, but this isn't standard C evaluation; usually, postfix is calculated as `5 + 35 = 40`.
4. **Explain and evaluate the following postfix expression:** `7 3 + 7 5 - 1`
   - **Answer:**
     - **Step 1:** Evaluate `7 3 +`, which gives `10`.
     - **Step 2:** Evaluate `7 5 -`, which gives `2`.
     - **Final Expression:** The result is `10 2`, but typically, postfix would be calculated as `10 - 2 = 8`.
5. **Explain the get node ( ) function in the context of singly linked list.**
   - **Answer:** The `getNode()` function in a singly linked list retrieves a node from the list at a specific position by traversing from the head node until the desired position is reached.
   - **Code Example in C:**

```
struct Node* getNode(struct Node* head, int position) {
    struct Node* current = head;
    int count = 0;
    while (current != NULL) {
        if (count == position)
            return current;
        count++;
        current = current->next;
    }
    return NULL;
}
```

Q.3 Attempt any 2 questions

(8 Marks)

1. **Write Push ( ) and Pop ( ) functions in a stack implemented using an array.**
   - **Push Function:**

```
void push(int stack[], int *top, int item, int maxSize) {
    if (*top == maxSize - 1) {
        printf("Stack Overflow\n");
        return;
    }
    stack[++(*top)] = item;
}
```

- **Pop Function:**

```
int pop(int stack[], int *top) {
    if (*top == -1) {
        printf("Stack Underflow\n");
        return -1;
    }
    return stack[(*top)--];
}
```

2. **Write Add ( ) and Remove ( ) functions in a Queue implemented using a linked list.**
   - **Add Function:**

```
void add(struct Queue* q, int value) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (q->rear == NULL) {
        q->front = q->rear = newNode;
        return;
    }
    q->rear->next = newNode;
    q->rear = newNode;
}
```

   - **Remove Function:**

```
int remove(struct Queue* q) {
    if (q->front == NULL) {
        printf("Queue Underflow\n");
        return -1;
    }
    struct Node* temp = q->front;
    int data = temp->data;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
```

```
    return data;
}
```

3. **Explain the cases of stack full and stack empty, while implementing a stack using an array.**
    - **Answer:**
        - **Stack Full:** A stack is full when the `top` index reaches `maxSize - 1`. At this point, no more elements can be added unless the array size is increased or elements are removed.
        - **Stack Empty:** A stack is empty when the `top` index is `-1`, meaning no elements are currently in the stack. Pop operations cannot be performed on an empty stack.

Q.4 (A) Explain with a suitable diagram, and code the function of Insert ( ) in a singly linked list. Assume the node contains an integer value and the nodes are arranged in an ascending order.

(5 Marks)

- **Explanation:** To insert a node into a singly linked list while maintaining ascending order, you need to find the correct position by traversing the list. Insert the new node before the first node that has a larger value or at the end if all existing nodes are smaller.
- **Code Example:**

```
struct Node* insert(struct Node* head, int value) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = value;
    if (head == NULL || head->data >= value) {
        newNode->next = head;
        return newNode;
    }
    struct Node* current = head;
    while (current->next != NULL && current->next->data < value) {
        current = current->next;
    }
    newNode->next = current->next;
    current->next = newNode;
    return head;
}
```

(B) Explain with suitable diagram, and code the function of Traverse ( ) in a singly linked list.

(5 Marks)

- **Explanation:** Traversing a singly linked list involves starting at the head node and moving through each subsequent node until the end of the list is reached.
- **Code Example:**

```
void traverse(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
```

OR

(B) Explain with suitable diagram and code the function of Append ( ) in a singly linked list.

(5 Marks)

- **Explanation:** The append function adds a new node to the end of the singly linked list. If the list is empty, the new node becomes the head. Otherwise, you traverse to the last node and set its `next` pointer to the new node.
- **Code Example:**

```
struct Node* append(struct Node* head, int value) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if (head == NULL) {
        return newNode;
    }
    struct Node* current = head;
    while (current->next != NULL) {
     current = current->next;
    }
    current->next = newNode;
return head;
}
```