

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  struct node {
4      int data;
5      struct node *next;
6  };
7  struct node *head = NULL;
8
9  void createList(int n) {
10 void insertAtBeginning(int data) {
41     struct node *newNode = (struct node*)malloc(sizeof(struct node));
42     if (newNode == NULL) {
43         printf("Memory allocation failed.\n");
44         return;
45     }
46     newNode->data = data;
47     newNode->next = head;
48     head = newNode;
49     printf("Node inserted at the beginning\n");
50 }
51 void insertAtEnd(int data) {
52     struct node *newNode = (struct node*)malloc(sizeof(struct node));
53     if (newNode == NULL) {
54         printf("Memory allocation failed.\n");
55         return;
56     }
57     newNode->data = data;
58     newNode->next = NULL;
59
60     if (head == NULL) {
61         head = newNode;
62     }
63     else {
64         struct node *temp = head;
65         while (temp->next != NULL)
66             temp = temp->next;
67         temp->next = newNode;
68     }
69     printf("Node inserted at the end\n");
70 }
71 void insertAtPosition(int data, int pos) {
72     int i;
73     struct node *newNode, *temp = head;
74
75     if (pos < 1) {
76         printf("Invalid position. Position must be 1 or greater.\n");
77         return;
78     }
79     if (pos == 1) {
80         insertAtBeginning(data);
81         return;
82     }
83     for (i = 1; i < pos - 1 && temp != NULL; i++)
84         temp = temp->next;
85     if (temp == NULL) {
86         printf("Position out of range: List is not long enough to reach position %d.\n", pos);
87         return;
88     }
89     newNode = (struct node*)malloc(sizeof(struct node));
90     if (newNode == NULL) {
91         printf("Memory allocation failed.\n");
92         return;
93     }
94     newNode->data = data;
95     newNode->next = temp->next;
96     temp->next = newNode;
97     printf("Node inserted at position %d\n", pos);
98 }
99 void displayList() {
100     struct node *temp = head;
101     if (head == NULL) {
102         printf("List is empty\n");
103         return;
104     }
105     printf("\nLinked list: ");
106     while (temp != NULL) {
107         printf("%d -> ", temp->data);
108         temp = temp->next;
109     }
110     printf("NULL\n");
111 }
112
113

```

```

114 int main() {
115     int choice, n, data, pos;
116
117     while (1) {
118         printf("\n--- Singly Linked List Operations ---\n");
119         printf("1. Create linked list\n");
120         printf("2. Insert at Beginning\n");
121         printf("3. Insert at any Position\n");
122         printf("4. Insert at End\n");
123         printf("5. Display list\n");
124         printf("6. Exit\n");
125         printf("Enter your choice: ");
126         if (scanf("%d", &choice) != 1) {
127
128             while (getchar() != '\n');
129             printf("Invalid input. Please enter a number.\n");
130             continue;
131         }
132
133         switch (choice) {
134             case 1:
135                 printf("Enter number of nodes: ");
136                 scanf("%d", &n);
137                 createList(n);
138                 break;
139             case 2:
140                 printf("Enter data to insert: ");
141                 scanf("%d", &data);
142                 insertAtBeginning(data);
143                 break;
144             case 3:
145                 printf("Enter data: ");
146                 scanf("%d", &data);
147                 printf("Enter position: ");
148                 scanf("%d", &pos);
149                 insertAtPosition(data, pos);
150                 break;
151             case 4:
152                 printf("Enter data to insert: ");
153                 scanf("%d", &data);
154                 insertAtEnd(data);
155                 break;
156             case 5:
157                 displayList();
158                 break;
159             case 6:
160                 printf("Exiting...\n");
161                 exit(0);
162             default:
163                 printf("Invalid choice. Try again.\n");
164         }
165     }
166     return 0;

```

```

---- Singly Linked List Operations ----
1. Create linked list
2. Insert at Beginning
3. Insert at any Position
4. Insert at End
5. Display list
6. Exit
Enter your choice: 1
Enter number of nodes: 3
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30

Linked list created successfully

---- Singly Linked List Operations ----
1. Create linked list
2. Insert at Beginning
3. Insert at any Position
4. Insert at End
5. Display list
6. Exit
Enter your choice: 2
Enter data to insert: 10
Node inserted at the beginning

---- Singly Linked List Operations ----
1. Create linked list
2. Insert at Beginning
3. Insert at any Position
4. Insert at End
5. Display list
6. Exit
Enter your choice: 3
Enter data: 4
Enter position: 3
Node inserted at position 3

---- Singly Linked List Operations ----
1. Create linked list
2. Insert at Beginning
3. Insert at any Position
4. Insert at End
5. Display list
6. Exit
Enter your choice: 4
Enter data to insert: 6
Node inserted at the end

---- Singly Linked List Operations ----
1. Create linked list
2. Insert at Beginning
3. Insert at any Position
4. Insert at End
5. Display list
6. Exit
Enter your choice: 6
Exiting...

Process returned 0 (0x0)   execution time : 41.343 s
Press any key to continue.

```

Lab Program-4  
Q. WAP to implement single linked list with the following operations

- Create a linked list
- Insertion of a node at
  - first position
  - any position
  - End position
- Display the contents of linked lists

Pseudo code :-

BEGIN

Define Structure Node with contents data, next pointer

Declare head pointer to NULL

Function CreateLinkedList(n)

Declare i, data, temp, newnode

For i ← 1 To n DO

Print "Enter value:"

Read value from input

newnode ← (struct Node\*) malloc (sizeof(struct Node))

newnode.data ← data

newnode.next ← NULL

if head == NULL

head ← temp ← newnode

Else

temp.next ← newnode

END IF

END Function

Function InsertAtBeginning(data)

newnode ← (struct Node\*) malloc (sizeof(struct Node))

newnode.data ← data

newnode.next ← head

head ← newnode

End function.

```

void InsertAtPosition (int data, int pos) {
    int i;
    struct node *temp = head;
    if (pos == 1) {
        printf("Invalid position, position must be 1 or greater\n");
        return;
    }
    if (pos == 1) {
        InsertAtBeginning(data);
        return;
    }
    for (i = 2; temp != NULL; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("position not found. List is not long enough to reach position %d.\n", pos);
        return;
    }
    struct node *newnode = malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }
    newnode->data = data;
    newnode->next = temp->next;
    temp->next = newnode;
    printf("Node inserted at position %d.\n", pos);
}

void displayList() {
    struct node *temp = head;
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Linked list: ");
    while (temp != NULL) {
        printf("%d-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

```

```

int main() {
    int choice, n, data, pos;
    while (1) {
        printf("\n... simply linked list operations ... \n");
        printf("1. Insert at beginning\n");
        printf("2. Insert at any position\n");
        printf("3. Insert at end\n");
        printf("4. Display List\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        if (scanf("%d", &choice) != 1) {
            while (getchar() != '\n');
            printf("Invalid input, please enter a number.\n");
            continue;
        }
        switch (choice) {
            case 1: printf("Enter number to insert: ");
                    scanf("%d", &n);
                    InsertAtBeginning(n); break;
            case 2: printf("Enter data to insert: ");
                    scanf("%d", &data);
                    printf("Enter position: ");
                    InsertAtPosition(data, pos); break;
            case 3: printf("Enter data: ");
                    scanf("%d", &data);
                    printf("Enter position: ");
                    scanf("%d", &pos);
                    InsertAtPosition(data, pos); break;
            case 4: printf("Enter data to insert: ");
                    scanf("%d", &data);
                    InsertAtEnd(data); break;
            case 5: displayList(); break;
            default: printf("Exiting... \n"); exit(0);
        }
        printf("Invalid choice. Try again.\n");
    }
    return 0;
}

```

```

Function InsertAtEnd (int data) {
    newnode = allocate memory;
    newnode->data = data;
    newnode->next = NULL;
    if (head == NULL) {
        head = newnode;
    }
    else {
        struct node *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newnode;
    }
    end function

Function InsertAtAnyPosition (int data, int pos) {
    newnode = allocate memory;
    newnode->data = data;
    if (pos == 1) {
        newnode->next = head;
        head = newnode;
        return;
    }
    temp = head;
    count = 1;
    while (temp->next != NULL and count < pos-1) {
        temp = temp->next;
        count++;
    }
    end while
    if (temp->next == NULL) {
        temp->next = newnode;
        newnode->next = NULL;
    }
    else {
        newnode->next = temp->next;
        temp->next = newnode;
    }
    end function

```

```

Code 2:
#include <stdio.h>
struct node {
    int data;
    struct node *next;
};

struct node *head = NULL;

void InsertAtBeginning (int data) {
    struct node *newnode = (struct node *) malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }
    newnode->data = data;
    newnode->next = head;
    head = newnode;
    printf("Node inserted at the beginning.\n");
}

void InsertAtEnd (int data) {
    struct node *newnode = (struct node *) malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }
    newnode->data = data;
    newnode->next = NULL;
    if (head == NULL) {
        head = newnode;
    }
    else {
        struct node *temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newnode;
    }
}

```



SLL - Singly Linked List Operations - ----

1. Create linked list
2. Insert at Beginning
3. Insert at any position
4. Insert at End
5. Display list
6. Exit

Enter your choice: 1  
 Enter number of nodes: 3  
 Enter data for node 1: 10  
 Enter data for node 2: 20  
 Enter data for node 3: 30  
 Linked list created Successfully.

----- Singly Linked List Created Operations -----

1. Create linked list
2. Insert at Beginning
3. Insert at any position
4. Insert at End
5. Display list
6. Exit

Enter your choice: 2  
 Enter data to insert: 10  
 Node inserted at the beginning.

----- Singly Linked List operations -----

1. Create linked list
2. Insert at Beginning
3. Insert at any position
4. Insert at End
5. Display list
6. Exit

Enter your choice: 3  
 Enter data: 4  
 Enter position: 3  
 Node inserted at position 3

Enter your choice: 4  
 Enter data to insert: 6  
 Node inserted at the End

Enter your choice: 5  
 Displaying

*[Handwritten signature]*

Lab 5  
 a) No. of  
 LL b) d  
 pseudocode