

PRG-6

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node *next;
};

// Global heads
struct Node *head1 = NULL;
struct Node *head2 = NULL;

// Function prototypes
struct Node* createList(int n);
void displayList(struct Node *head);
void sortLinkedList(struct Node *head);
struct Node* reverseLinkedList(struct Node *head);
struct Node* concatenateLinkedList(struct Node *head1, struct Node *head2);

// Create list
struct Node* createList(int n) {
    struct Node *head = NULL, *newNode, *temp;
    int data;

    if (n <= 0) {
        printf("Number of nodes should be greater than 0\n");
        return NULL;
    }

    newNode = (struct Node*) malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        return NULL;
    }

    temp = head;
    head = newNode;
    head->data = 1;
    head->next = NULL;

    for (int i = 2; i < n; i++) {
        newNode = (struct Node*) malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed\n");
            free(head);
            return NULL;
        }

        newNode->data = i;
        newNode->next = head;
        head = newNode;
    }
}
```

```
for (int i = 1; i <= n; i++) {  
    newNode = (struct Node*)malloc(sizeof(struct Node));  
    if (newNode == NULL) {  
        printf("Memory allocation failed\n");  
        return head;  
    }  
  
    printf("Enter data for node %d: ", i);  
    scanf("%d", &data);  
  
    newNode->data = data;  
    newNode->next = NULL;  
  
    if (head == NULL)  
        head = newNode;  
    else  
        temp->next = newNode;  
  
    temp = newNode;  
}  
  
printf("Linked list created successfully\n");  
return head;  
}  
  
// Display  
void displayList(struct Node *head) {  
    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");

}

// Sort
void sortLinkedList(struct Node *head) {
    struct Node *i, *j;
    int tempData;

    if (head == NULL) {
        printf("List is empty, cannot sort.\n");
        return;
    }

    for (i = head; i->next != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                tempData = i->data;
                i->data = j->data;
                j->data = tempData;
            }
        }
    }
}

```

```
printf("Linked list sorted successfully\n");

}

// Reverse

struct Node* reverseLinkedList(struct Node *head) {

    struct Node *prev = NULL, *curr = head, *next = NULL;

    while (curr != NULL) {

        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }

    printf("Linked list reversed successfully\n");
    return prev;
}

// Concatenate

struct Node* concatenateLinkedList(struct Node *head1, struct Node *head2) {

    struct Node *temp;

    if (head1 == NULL)
        return head2;

    temp = head1;
    while (temp->next != NULL)
        temp = temp->next;

    temp->next = head2;
```

```
printf("Linked lists concatenated successfully\n");

return head1;

}

// Menu

int main() {

    int choice, n;

    while (1) {

        printf("\n=====\\n");
        printf("      LINKED LIST MENU      \\n");
        printf("=====\\n");
        printf("1. Create List 1\\n");
        printf("2. Create List 2\\n");
        printf("3. Display List 1\\n");
        printf("4. Display List 2\\n");
        printf("5. Sort List 1\\n");
        printf("6. Reverse List 1\\n");
        printf("7. Concatenate List 1 + List 2\\n");
        printf("8. Exit\\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter number of nodes for List 1: ");
                scanf("%d", &n);
                head1 = createList(n);
                break;
        }
    }
}
```

case 2:

```
printf("Enter number of nodes for List 2: ");
scanf("%d", &n);
head2 = createList(n);
break;
```

case 3:

```
displayList(head1);
break;
```

case 4:

```
displayList(head2);
break;
```

case 5:

```
sortLinkedList(head1);
break;
```

case 6:

```
head1 = reverseLinkedList(head1);
break;
```

case 7:

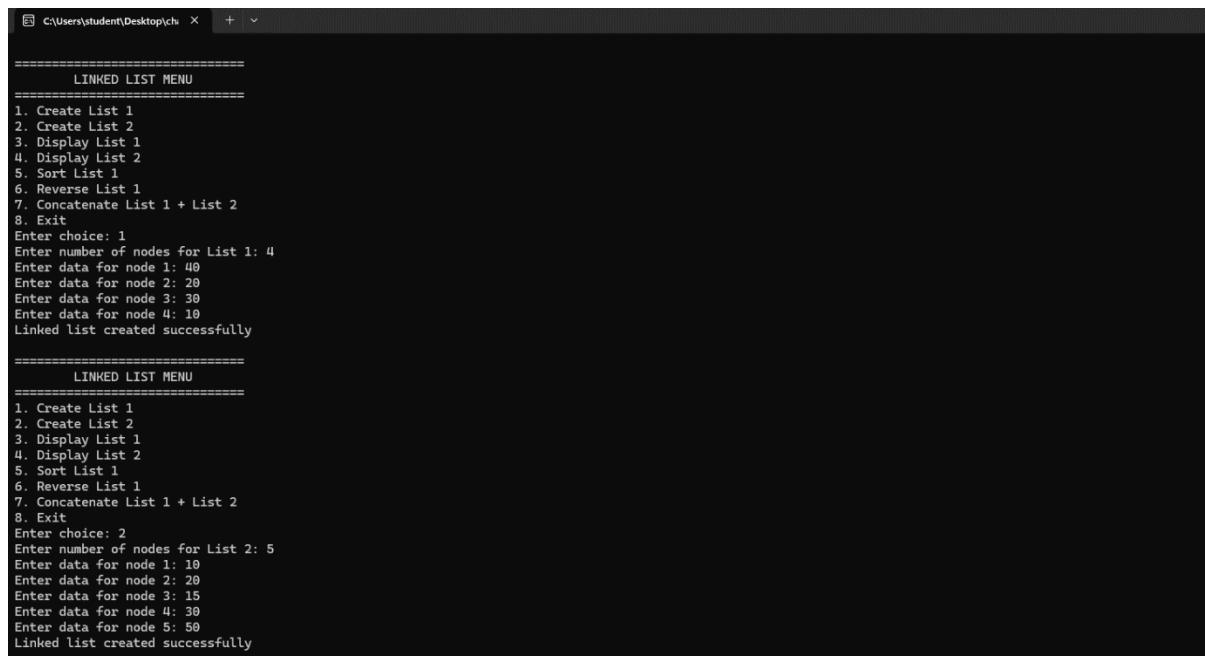
```
head1 = concatenateLinkedList(head1, head2);
printf("After concatenation:\n");
displayList(head1);
break;
```

case 8:

```
printf("Exiting program...\n");
exit(0);
```

```
default:  
    printf("Invalid choice. Try again.\n");  
}  
}  
  
return 0;  
}
```

OUTPUT:



The screenshot shows a terminal window titled 'C:\Users\student\Desktop\chv' with the following content:

```
=====  
LINKED LIST MENU  
=====  
1. Create List 1  
2. Create List 2  
3. Display List 1  
4. Display List 2  
5. Sort List 1  
6. Reverse List 1  
7. Concatenate List 1 + List 2  
8. Exit  
Enter choice: 1  
Enter number of nodes for List 1: 4  
Enter data for node 1: 40  
Enter data for node 2: 20  
Enter data for node 3: 30  
Enter data for node 4: 10  
Linked list created successfully  
=====  
LINKED LIST MENU  
=====  
1. Create List 1  
2. Create List 2  
3. Display List 1  
4. Display List 2  
5. Sort List 1  
6. Reverse List 1  
7. Concatenate List 1 + List 2  
8. Exit  
Enter choice: 2  
Enter number of nodes for List 2: 5  
Enter data for node 1: 10  
Enter data for node 2: 20  
Enter data for node 3: 15  
Enter data for node 4: 30  
Enter data for node 5: 50  
Linked list created successfully
```

```
C:\Users\student\Desktop\ch1> + ▾

=====
LINKED LIST MENU
=====
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 3
Linked List: 40 -> 20 -> 30 -> 10 -> NULL

=====
LINKED LIST MENU
=====
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 4
Linked List: 10 -> 20 -> 15 -> 30 -> 50 -> NULL

=====
LINKED LIST MENU
=====
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 5
Linked list sorted successfully
```

```
C:\Users\student\Desktop\ch1 + ▾

=====
LINKED LIST MENU
=====
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 3
Linked List: 10 -> 20 -> 30 -> 40 -> NULL

=====
LINKED LIST MENU
=====
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 6
Linked list reversed successfully

=====
LINKED LIST MENU
=====
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 3
Linked List: 40 -> 30 -> 20 -> 10 -> NULL
```

```
C:\Users\student\Desktop\ch1> + | v

=====
LINKED LIST MENU
=====
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 7
Linked lists concatenated successfully
After concatenation:
Linked List: 40 -> 30 -> 20 -> 10 -> 10 -> 20 -> 15 -> 30 -> 50 -> NULL

=====
LINKED LIST MENU
=====
1. Create List 1
2. Create List 2
3. Display List 1
4. Display List 2
5. Sort List 1
6. Reverse List 1
7. Concatenate List 1 + List 2
8. Exit
Enter choice: 8
Exiting program...

Process returned 0 (0x0) execution time : 60.018 s
Press any key to continue.
|
```

OBSERVATION:

11/11/25

Date _____
Page _____

Programs :-

- a. NAP to implement Singly Linked List with following operations : sort the linked list, Reverse the linked list, Concatenation of two linked list.

Pseudo

Ans To sort a linked list (Bubble Sort) :-

- 1 Start with the first node as pointer i.
- 2 Repeat while i is not the last node:
 - Set pointer j to the node after i.
 - Repeat while j is not NULL:
 - If the data in i is greater than the data in j,
swap the data using temp.
 - Move j to the next data.
 - Move i to the next node.

To reverse a linked list :-

- 1 Set prev to NULL.
- 2 Set curr to the head of the list.
- 3 Repeat while curr is not NULL:
 - Store the next node of curr in nextNode.
 - Change the next pointer of curr to prev.
 - Move prev to curr.
 - Move curr to nextNode.
- 4 After the loop, set the head of the list to prev.

To concatenate two linked list :-

1. If the first list is empty, make the second list as the resulting list.
2. Otherwise, start from head of the first list.
3. Move through the list until the last node is reached.
4. Set the next pointer of the last node of the first list to the head of the second list.

\Rightarrow `#include <stdio.h>`
`#include <stdlib.h>`

~~struct Node {~~
~~int data;~~
~~struct Node *next;~~
};

~~struct Node *head1 = NULL;~~

~~struct Node *head2 = NULL;~~

~~struct Node *CreateList (int n);~~
~~void displayList (struct Node *head);~~
~~void sortlinkedList (struct Node *head);~~
~~struct Node *reverseLinkedList (struct Node *head);~~
~~struct Node *concatenateLinkedList (struct Node *head1,~~

~~struct Node *head2);~~
~~struct Node *createList (int n);~~
~~struct Node *head3 = NULL; struct Node *newNode, *temp;~~
~~int data;~~

```
if (n <= 0) {  
    printf ("Number of nodes should be  
            greater than 0\n");  
    return NULL;  
}
```

```
for (int i = 1; i < n; i++) {  
    newNode = (struct Node*) malloc (sizeof (struct  
                                         Node));  
    if (newNode == NULL) {  
        printf ("Memory allocation failed\n");  
        return head;  
    }
```

```
    printf ("Enter data for node %d: ", i);  
    scanf ("%d", &data);
```

```
    newNode->data = data;  
    newNode->next = NULL;
```

```
    if (head == NULL)  
        head = newNode;
```

else

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

✓ temp = newNode;

}

↳

```
    printf ("Linked List created successfully\n");  
    return head;
```

```
void displayList (struct Node *head) {
    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");
}
```

```
void Sortlinkedlist (struct Node *head) {
    struct Node *i, *j;
    int tempData;
    if (head == NULL) {
        printf ("List is empty, cannot sort. \n");
        return;
    }
    for (i = head; i->next != NULL; i = i->next)
        for (j = i->next; j != NULL; j = j->next)
            if (i->data > j->data) {
                tempData = i->data;
                i->data = j->data;
                j->data = tempData;
            }
}
```

{ j->data = tempData;

}

{ }

printf ("Linked List sorted successfully\n");

3 struct Node * reverseLinkedList (struct Node *head)

{

 struct Node * prev = NULL, * curr = head,
 * next = NULL;

 while (curr != NULL) {

 next = curr->next;

 curr->next = prev;

 prev = curr;

 curr = next;

}

 printf ("Linked List reversed successfully\n");

 return prev;

}

3 struct Node * concatenateLinkedList (struct
 Node *head1, struct Node *head2)

 if (head1 == NULL)

 return head2;

 temp = head1;

 while (temp->next != NULL)

 temp = temp->next;

```
temp->next = head;
printf ("Linked lists concatenated
successfully\n");
return head;
```

```
}
```

```
int main () {
    int choice, n;
    while (1) {
        printf ("\n===== \n");
        printf ("1. Linked List Menu \n");
        printf ("2. Create List 1 \n");
        printf ("3. Display List 1 \n");
        printf ("4. Display List 2 \n");
        printf ("5. Sort List 1 \n");
        printf ("6. Reverse List 1 \n");
        printf ("7. Concatenate List 1 + List 2 \n");
        printf ("8. Exit \n");
        printf ("Enter choice: ");
        scanf ("%d", &choice);
    }
}
```

```
switch (choice) {
    case 1:
        printf ("Enter no. of nodes for
List 1: ");
        scanf ("%d", &n);
        head_1 = createList (n);
        break;
```

case 2:

```
printf ("Enter number of nodes for  
List 2 : ");  
scanf ("%d", &n);  
head2 = createList(n);  
break;
```

case 3:

```
displayList(head1);  
break;
```

case 4:

```
displayList(head2);  
break;
```

case 5:

```
sortLinkedList(head1);  
break;
```

case 6:

```
head1 = reverseLinkedList(head1);  
break;
```

case 7:

```
head1 = concatenateLinkedList(head1, head2);  
printf ("After concatenation: \n");  
displayList(head1);  
break;
```

case 8:

```
printf ("Exiting program....\n");  
exit (0);
```

```
    default;  
    { printf ("Invalid choice. Try again\n");  
    }  
    return 0;  
}
```

Output:

Linked List Menu

- 1. Create List 1
- 2. Create List 2
- 3. Display List 1
- 4. Display List 2
- 5. Sort List 1
- 6. Reverse List 1
- 7. Concatenate List 1 + List 2
- 8. Exit.

Enter choice: 1

Enter number of nodes for List 1: 4

Enter data for node 1: 40

Enter data for node 2: 20

Enter data for node 3: 30

Enter data for node 4: 10

Linked List created successfully.

Enter choice: 2

Enter number of nodes for List 2: 5

Enter data for node 1: 10

Enter data for node 2: 20

Enter data for node 3: 35

Enter data for node 4 : 30

Enter data for node 5 : 50

Linked List created successfully.

Enter choice : 3

Linked List : 40 → 20 → 30 → 10 → NULL

Enter choice : 4

Linked List : 10 → 20 → 15 → 30 → 50 → NULL

Enter choice : 5

Linked List sorted successfully.

Enter choice : 3

Linked List : 10 → 20 → 30 → 40 → NULL

Enter choice : 6

Linked List reversed successfully.

Enter choice : 3

Linked List : 40 → 30 → 20 → 10 → NULL

Enter choice : 7

Linked List concatenated successfully

After concatenation:

Linked List : 40 → 30 → 20 → 10 → 10 → 20 → 15 → 30 → 50 → NULL.

8

Enter choice : 8

Exiting program....

