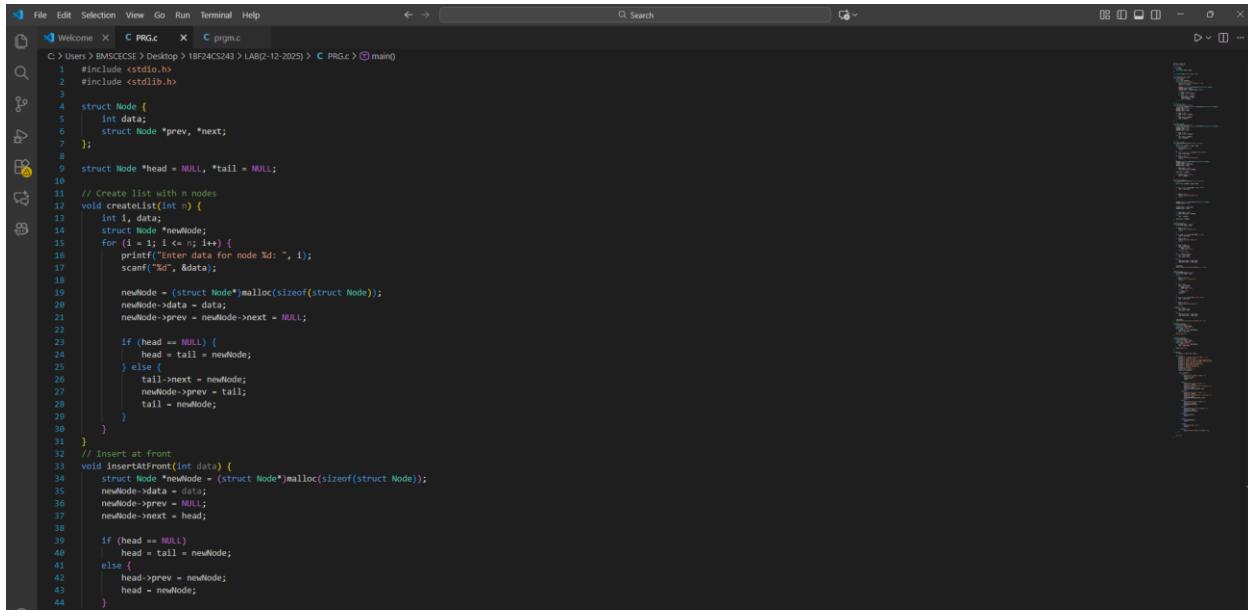
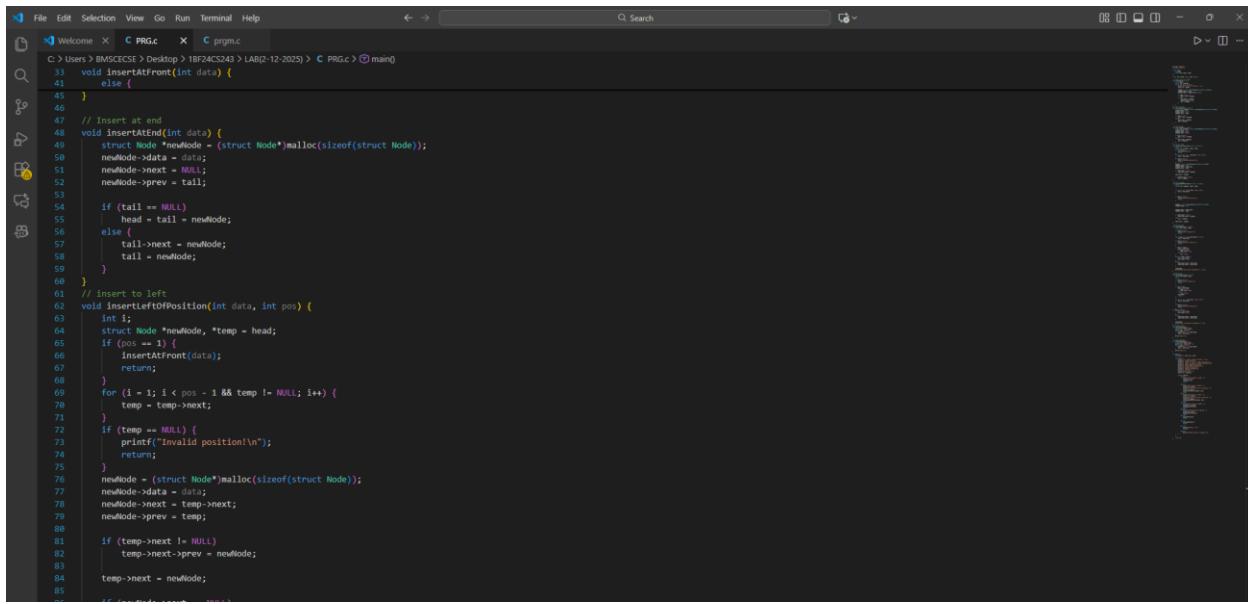


LAB(2-12-2025)

INPUT



```
C:\Users\BMSCECSE\Desktop>1BF24CS243>LAB(2-12-2025)>C PRG.c>(main)
1 #include <cslib.h>
2 #include <stdlib.h>
3
4 struct Node {
5     int data;
6     struct Node *prev, *next;
7 };
8
9 struct Node *head = NULL, *tail = NULL;
10
11 // Create list with n nodes
12 void createlist(int n) {
13     int i, data;
14     struct Node *newNode;
15     for (i = 1; i <= n; i++) {
16         printf("Enter data for node %d: ", i);
17         scanf("%d", &data);
18
19         newNode = (struct Node*)malloc(sizeof(struct Node));
20         newNode->data = data;
21         newNode->prev = NULL;
22         newNode->next = NULL;
23
24         if (head == NULL) {
25             head = tail = newNode;
26         } else {
27             tail->next = newNode;
28             newNode->prev = tail;
29             tail = newNode;
30         }
31     }
32
33 // Insert at front
34 void insertAtFront(int data) {
35     struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
36     newNode->data = data;
37     newNode->prev = NULL;
38     newNode->next = head;
39
40     if (head == NULL) {
41         head = tail = newNode;
42     } else {
43         head->prev = newNode;
44         head = newNode;
45     }
46 }
```



```
33 void insertAtFront(int data) {
34     struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
35     newNode->data = data;
36     newNode->next = NULL;
37     newNode->prev = tail;
38
39     if (tail == NULL)
40         head = tail = newNode;
41     else {
42         tail->next = newNode;
43         tail = newNode;
44     }
45 }
46
47 // Insert at end
48 void insertAtEnd(int data) {
49     struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
50     newNode->data = data;
51     newNode->next = NULL;
52     newNode->prev = tail;
53
54     if (tail == NULL)
55         head = tail = newNode;
56     else {
57         tail->next = newNode;
58         tail = newNode;
59     }
60 }
61
62 // insert to left of position
63 void insertToLeftPosition(int data, int pos) {
64     int i;
65     struct Node *newNode, *temp = head;
66     if (pos == 1) {
67         insertAtFront(data);
68         return;
69     }
70     for (i = 1; i < pos - 1 && temp != NULL; i++) {
71         temp = temp->next;
72     }
73     if (temp == NULL) {
74         printf("Invalid position!\n");
75         return;
76     }
77     newNode = (struct Node*)malloc(sizeof(struct Node));
78     newNode->data = data;
79     newNode->next = temp->next;
80     newNode->prev = temp;
81
82     if (temp->next != NULL)
83         temp->next->prev = newNode;
84
85     temp->next = newNode;
86 }
```

The screenshot shows a code editor interface with two tabs: `PRG.c` and `prgm.c`. The `PRG.c` tab is currently active, displaying the following C code:

```
1 //insert at left
2 void insertLeftOfPosition(int data, int pos) {
3     struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
4     newNode->data = data;
5     newNode->next = temp->next;
6     newNode->prev = temp;
7
8     if (temp->next == NULL)
9         temp->next->prev = newNode;
10    temp->next = newNode;
11
12    if (newNode->next == NULL)
13        tail = newNode;
14
15 }
16
17 //insert at right
18 void insertRightOfPosition(int data, int pos) {
19     int i;
20     struct Node *newNode, *temp = head;
21
22     for (i = 1; i < pos && temp != NULL; i++) {
23         temp = temp->next;
24     }
25
26     if (temp == NULL) {
27         printf("invalid position!\n");
28         return;
29     }
30
31     newNode = (struct Node*)malloc(sizeof(struct Node));
32     newNode->data = data;
33
34     newNode->next = temp->next;
35     newNode->prev = temp;
36
37     if (temp->next != NULL)
38         temp->next->prev = newNode;
39     else
40
41 }
```

The `prgm.c` tab contains the same code with additional comments and a `deletedByValue` function:

```
1 //insert at left
2 void insertLeftOfPosition(int data, int pos) {
3     struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
4     newNode->data = data;
5     newNode->next = temp->next;
6     newNode->prev = temp;
7
8     if (temp->next != NULL)
9         temp->next->prev = newNode;
10    else
11        tail = newNode;
12
13    temp->next = newNode;
14
15 }
16
17 //insert at right
18 void insertRightOfPosition(int data, int pos) {
19     int i;
20     struct Node *newNode, *temp = head;
21
22     for (i = 1; i < pos && temp != NULL; i++) {
23         temp = temp->next;
24     }
25
26     if (temp == NULL) {
27         printf("invalid position!\n");
28         return;
29     }
30
31     newNode = (struct Node*)malloc(sizeof(struct Node));
32     newNode->data = data;
33
34     newNode->next = temp->next;
35     newNode->prev = temp;
36
37     if (temp->next != NULL)
38         temp->next->prev = newNode;
39     else
40        tail = newNode;
41
42
43 //delete by value
44 void deletedByValue(int value) {
45     struct Node *temp = head;
46
47     if (head == NULL) {
48         printf("list is empty!\n");
49         return;
50     }
51
52     while (temp != NULL && temp->data != value)
53         temp = temp->next;
54
55     if (temp == NULL) {
56         printf("Value not found!\n");
57         return;
58     }
59
60     if (temp == head) {
61         head = head->next;
62         if (head != NULL)
63             head->prev = NULL;
64         else
65             tail = NULL;
66     }
67     else if (temp == tail) {
68         tail = tail->prev;
69         tail->next = NULL;
70     }
71     else {
72         temp->prev->next = temp->next;
73     }
74 }
```

```
C > Users > BMSCCSE > Desktop > 1B24CS243 > LAB02-12-2025 > C PRG.c > (main)
124 void deleteByValue(int value) {
125     struct Node *temp = head;
126     struct Node *tail = temp;
127 
128     while (temp != NULL) {
129         if (temp->data == value) {
130             if (temp == head) {
131                 head = temp->next;
132                 if (head != NULL)
133                     head->prev = NULL;
134                 else
135                     tail = head;
136             }
137             else if (temp == tail) {
138                 temp->prev->next = NULL;
139                 tail = temp->prev;
140             }
141             else {
142                 temp->prev->next = temp->next;
143                 temp->next->prev = temp->prev;
144             }
145             free(temp);
146             printf("Node with value %d deleted.\n", value);
147         }
148         temp = temp->next;
149     }
150 }
151 
152 // Delete by pos
153 void deleteAtPosition(int pos) {
154     struct Node *temp = head;
155     int i;
156 
157     if (head == NULL) {
158         printf("List is empty!\n");
159         return;
160     }
161 
162     if (pos == 1) {
163         head = head->next;
164         if (head != NULL)
165             head->prev = NULL;
166         else
167             tail = NULL;
168         free(temp);
169         return;
170     }
171     for (i = 1; i < pos && temp != NULL; i++) {
172         temp = temp->next;
173     }
174 
175     if (temp == NULL) {
176         printf("Invalid position!\n");
177         return;
178     }
179     if (temp == tail) {
180         tail = tail->prev;
181     }
182     else {
183         temp->prev->next = temp->next;
184         temp->next->prev = temp->prev;
185     }
186     free(temp);
187     printf("Node at position %d deleted.\n", pos);
188 }
189 
190 // Display forward
191 void displayForward() {
192     struct Node *temp = head;
193     printf("List (Forward): ");
194     while (temp != NULL) {
195         printf("%d <-> ", temp->data);
196         temp = temp->next;
197     }
198     printf("NULL\n");
199 }
200 
201 // Display backward
202 void displayBackward() {
203     struct Node *temp = tail;
204     printf("List (Backward): ");
205     while (temp != NULL) {
206         printf("%d <-> ", temp->data);
207         temp = temp->prev;
208     }
209     printf("NULL\n");
210 }
211 
212 int main() {
213     int choice, n, data, pos, value;
214 }
```

The code implements a doubly linked list with nodes having data, prev, and next pointers. The `deleteByValue` function removes a node with a specific data value. The `deleteAtPosition` function removes the node at a specified position. Both functions handle cases where the list is empty or has only one node. The `displayForward` and `displayBackward` functions print the list's contents in forward and backward directions respectively.

```
222 int main() {
223     int choice, n, data, pos, value;
224
225     while (1) {
226         printf("\n--- Doubly Linked List Menu ---\n");
227         printf("1. Create List\n");
228         printf("2. Insert to Left of a Node (Position)\n");
229         printf("3. Insert to right of a Node (Position)\n");
230         printf("4. Delete Node by Value\n");
231         printf("5. Delete Node by position\n");
232         printf("6. Display Forward\n");
233         printf("7. Display Backward\n");
234         printf("8. Exit\n");
235         printf("Enter choice: ");
236         scanf("%d", &choice);
237
238         switch (choice) {
239             case 1:
240                 printf("Enter number of nodes: ");
241                 scanf("%d", &n);
242                 createlist(n);
243                 break;
244
245             case 2:
246                 printf("Enter data to insert: ");
247                 scanf("%d", &data);
248                 printf("Insert to the left of position: ");
249                 scanf("%d", &pos);
250                 insertLeftToPosition(data, pos);
251                 break;
252
253             case 3:
254                 printf("Enter data to insert: ");
255                 scanf("%d", &data);
256                 printf("Insert to the left of position: ");
257                 scanf("%d", &pos);
258                 insertRightToPosition(data, pos);
259                 break;
260
261             case 4:
262                 printf("Enter value to delete: ");
263                 scanf("%d", &value);
264                 deletebyValue(value);
265                 break;
266
267             case 5:
268                 printf("Enter position to delete: ");
269                 scanf("%d", &pos);
270                 deletedAtPosition(pos);
271                 break;
272
273             case 6:
274                 displayForward();
275                 break;
276
277             case 7:
278                 displayBackward();
279                 break;
280
281             case 8:
282                 printf("Exiting...\n");
283                 exit(0);
284
285             default:
286                 printf("Invalid choice! Try again.\n");
287         }
288     }
289
290     return 0;
291 }
```

OUTPUT

```
File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\BPSCECS\Desktop\1BF24CS243\LAB(2-12-2025)\output>
PS C:\Users\BPSCECS\Desktop\1BF24CS243\LAB(2-12-2025)\output> & .\VFLG.exe

... Doubly Linked List Menu ...
1. Create List
2. Insert to Left of a Node (Position)
3. Insert to right of a Node (Position)
4. Delete Node by Value
5. Delete Node by position
6. Display forward
7. Display backward
8. Exit
Enter choice: 1
Enter number of nodes: 2
Enter data for node 1: 10
Enter data for node 2: 20
... Doubly Linked List Menu ...
1. Create List
2. Insert to Left of a Node (Position)
3. Insert to right of a Node (Position)
4. Delete Node by Value
5. Delete Node by position
6. Display forward
7. Display backward
8. Exit
Enter choice: 2
Enter data to insert: 30
Insert to the left of position: 2
... Doubly Linked List Menu ...
1. Create List
2. Insert to Left of a Node (Position)
3. Insert to right of a Node (Position)
4. Delete Node by Value
5. Delete Node by position
6. Display forward
7. Display backward
8. Exit
Enter choice: 3
Enter data to insert: 40
Insert to the left of position: 2
... Doubly Linked List Menu ...
1. Create List
2. Insert to Left of a Node (Position)
3. Insert to right of a Node (Position)
4. Delete Node by Value
5. Delete Node by position
6. Display forward
7. Display backward
8. Exit
Enter choice: 4
Enter value to delete: 20
Node with value 20 Deleted.
```

```
File Edit Selection View Go Run Terminal Help
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Enter value to delete: 20
Node with value 20 deleted.
... Doubly Linked List Menu ...
1. Create List
2. Insert to Left of a Node (Position)
3. Insert to right of a Node (Position)
4. Delete Node by Value
5. Delete Node by position
6. Display forward
7. Display backward
8. Exit
Enter choice: 5
Enter position to delete: 3
Node at position 3 deleted.
... Doubly Linked List Menu ...
1. Create List
2. Insert to Left of a Node (Position)
3. Insert to right of a Node (Position)
4. Delete Node by Value
5. Delete Node by position
6. Display forward
7. Display backward
8. Exit
Enter choice: 6
List (forward): 10 <-> 30 <-> NULL
... Doubly Linked List Menu ...
1. Create List
2. Insert to Left of a Node (Position)
3. Insert to right of a Node (Position)
4. Delete Node by Value
5. Delete Node by position
6. Display forward
7. Display backward
8. Exit
Enter choice: 7
List (backward): 30 <-> 10 <-> NULL
... Doubly Linked List Menu ...
1. Create List
2. Insert to Left of a Node (Position)
3. Insert to right of a Node (Position)
4. Delete Node by Value
5. Delete Node by position
6. Display forward
7. Display backward
8. Exit
Enter choice: 8
Exiting...
PS C:\Users\BPSCECS\Desktop\1BF24CS243\LAB(2-12-2025)\output>
```

Ln 257, Col 17 Spaces: 4 UTF-8 CRLF { } C Signed out Win32