

WEEK-07

```
Start here x singlylist.c x SLL.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // Define a node structure
5  struct Node {
6      int data;
7      struct Node* next;
8  };
9
10 // Function to create a new node
11 struct Node* createNode(int data) {
12     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
13     if (!newNode) {
14         printf("Memory allocation failed!\n");
15         exit(1);
16     }
17     newNode->data = data;
18     newNode->next = NULL;
19     return newNode;
20 }
21
22 // Function to insert a new element at the beginning
23 void insertAtFirst(struct Node** head, int data) {
24     struct Node* newNode = createNode(data);
25     newNode->next = *head;
26     *head = newNode;
27 }
28
29 // Function to insert a new element at the end
30 void insertAtEnd(struct Node** head, int data) {
31     struct Node* newNode = createNode(data);
32     if (*head == NULL) {
33         *head = newNode;
34         return;
35     }
36     struct Node* temp = *head;
37     while (temp->next != NULL) {
38         temp = temp->next;
39     }
40     temp->next = newNode;
41 }
42
```

```

42
43 // Function to insert a new element at a specific position
44 void insertAtPosition(struct Node** head, int data, int position) {
45     if (position == 0) {
46         insertAtFirst(head, data);
47         return;
48     }
49     struct Node* newNode = createNode(data);
50     struct Node* temp = *head;
51     for (int i = 0; temp != NULL && i < position - 1; i++) {
52         temp = temp->next;
53     }
54     if (temp == NULL) {
55         printf("Position out of range\n");
56         free(newNode);
57         return;
58     }
59     newNode->next = temp->next;
60     temp->next = newNode;
61 }
62
63 // Function to delete a node from the beginning
64 void deleteAtFirst(struct Node** head) {
65     if (*head == NULL) {
66         printf("List is empty, nothing to delete.\n");
67         return;
68     }
69     struct Node* temp = *head;
70     *head = (*head)->next;
71     free(temp);
72     printf("Node deleted from the beginning.\n");
73 }
74

```

```

76 void deleteAtEnd(struct Node** head) {
77     if (*head == NULL) {
78         printf("List is empty, nothing to delete.\n");
79         return;
80     }
81     struct Node* temp = *head;
82     if (temp->next == NULL) { // Only one node in the list
83         free(temp);
84         *head = NULL;
85         printf("Node deleted from the end.\n");
86         return;
87     }
88     while (temp->next != NULL && temp->next->next != NULL) {
89         temp = temp->next;
90     }
91     free(temp->next);
92     temp->next = NULL;
93     printf("Node deleted from the end.\n");
94 }
95
96 // Function to delete a node from a specific position
97 void deleteAtPosition(struct Node** head, int position) {
98     if (*head == NULL) {
99         printf("List is empty, nothing to delete.\n");
100         return;
101     }
102     if (position == 0) {
103         deleteAtFirst(head);
104         return;
105     }
106     struct Node* temp = *head;
107     for (int i = 0; temp != NULL && i < position - 1; i++) {
108         temp = temp->next;
109     }
110     if (temp == NULL || temp->next == NULL) {
111         printf("Position out of range\n");
112         return;
113     }
114     struct Node* nodeToDelete = temp->next;
115     temp->next = temp->next->next;
116     free(nodeToDelete);
117     printf("Node deleted from position %d.\n", position);
118 }

```

```

119
120 // Function to print the linked list
121 void print(struct Node* head) {
122     if (head == NULL) {
123         printf("List is empty\n");
124         return;
125     }
126     struct Node* temp = head;
127     while (temp != NULL) {
128         printf("%d -> ", temp->data);
129         temp = temp->next;
130     }
131     printf("NULL\n");
132 }
133
134 // Function to sort the linked list (Bubble Sort)
135 void sortList(struct Node** head) {
136     if (*head == NULL) {
137         return;
138     }
139     struct Node *i, *j;
140     int temp;
141     for (i = *head; i != NULL; i = i->next) {
142         for (j = i->next; j != NULL; j = j->next) {
143             if (i->data > j->data) {
144                 temp = i->data;
145                 i->data = j->data;
146                 j->data = temp;
147             }
148         }
149     }
150     printf("List sorted.\n");
151 }
152

```

```

152
153 // Function to reverse the linked list
154 void reverseList(struct Node** head) {
155     struct Node* prev = NULL;
156     struct Node* current = *head;
157     struct Node* next = NULL;
158     while (current != NULL) {
159         next = current->next;
160         current->next = prev;
161         prev = current;
162         current = next;
163     }
164     *head = prev;
165     printf("List reversed.\n");
166 }
167
168 // Function to concatenate two linked lists
169 void concatenateLists(struct Node** head1, struct Node* head2) {
170     if (*head1 == NULL) {
171         *head1 = head2;
172         return;
173     }
174     struct Node* temp = *head1;
175     while (temp->next != NULL) {
176         temp = temp->next;
177     }
178     temp->next = head2;
179     printf("Lists concatenated.\n");
180 }
181

```

```

Start here x singlylist.c x SLL.c x
182 // Main function
183 int main() {
184     struct Node* head1 = NULL;
185     struct Node* head2 = NULL;
186     int choice, data, position;
187
188     while (1) {
189         printf("\n----- Singly Linked List ----- \n");
190         printf("1. Insert at beginning (List 1)\n");
191         printf("2. Insert at end (List 1)\n");
192         printf("3. Insert at position (List 1)\n");
193         printf("4. Insert at beginning (List 2)\n");
194         printf("5. Insert at end (List 2)\n");
195         printf("6. Insert at position (List 2)\n");
196         printf("7. Display list (List 1)\n");
197         printf("8. Sort list (List 1)\n");
198         printf("9. Reverse list (List 1)\n");
199         printf("10. Concatenate lists\n");
200         printf("11. Exit\n");
201         printf("Enter your choice: ");
202         scanf("%d", &choice);
203
204         switch (choice) {
205             case 1:
206                 printf("Enter data to insert: ");
207                 scanf("%d", &data);
208                 insertAtFirst(&head1, data);
209                 break;
210             case 2:
211                 printf("Enter data to insert: ");
212                 scanf("%d", &data);
213                 insertAtEnd(&head1, data);
214                 break;
215             case 3:
216                 printf("Enter data to insert: ");
217                 scanf("%d", &data);
218                 printf("Enter position (starting from 0): ");
219                 scanf("%d", &position);
220                 insertAtPosition(&head1, data, position);
221                 break;

```

```

222     case 4:
223         printf("Enter data to insert: ");
224         scanf("%d", &data);
225         insertAtFirst(&head2, data);
226         break;
227     case 5:
228         printf("Enter data to insert: ");
229         scanf("%d", &data);
230         insertAtEnd(&head2, data);
231         break;
232     case 6:
233         printf("Enter data to insert: ");
234         scanf("%d", &data);
235         printf("Enter position (starting from 0): ");
236         scanf("%d", &position);
237         insertAtPosition(&head2, data, position);
238         break;
239     case 7:
240         printf("List 1: ");
241         print(head1);
242         break;
243     case 8:
244         sortList(&head1);
245         break;
246     case 9:
247         reverseList(&head1);
248         break;
249     case 10:
250         concatenateLists(&head1, head2);
251         break;
252     case 11:
253         exit(0); // Exit the program
254     default:
255         printf("Invalid choice! Please try again.\n");
256     }
257 }
258 return 0;
259 }
260

```

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 1

Enter data to insert: 10

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 2

Enter data to insert: 50

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 1

Enter data to insert: 20

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

1. Insert at beginning (List 1)

Enter your choice: 1
Enter data to insert: 30

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 1
Enter data to insert: 40

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 7
List 1: 40 -> 30 -> 20 -> 10 -> 50 -> NULL

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 4
Enter data to insert: 90

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

Enter your choice: 5

Enter data to insert: 100

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 5

Enter data to insert: 200

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 7

List 1: 40 -> 30 -> 20 -> 10 -> 50 -> NULL

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 8

List sorted.

Enter your choice: 7

List 1: 10 -> 20 -> 30 -> 40 -> 50 -> NULL

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 9

List reversed.

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 7

List 1: 50 -> 40 -> 30 -> 20 -> 10 -> NULL

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 10

Lists concatenated.

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 7

List 1: 50 -> 40 -> 30 -> 20 -> 10 -> 90 -> 100 -> 200 -> NULL

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

1. Insert at beginning (List 1)
2. Insert at end (List 1)
3. Insert at position (List 1)
4. Insert at beginning (List 2)
5. Insert at end (List 2)
6. Insert at position (List 2)
7. Display list (List 1)
8. Sort list (List 1)
9. Reverse list (List 1)
10. Concatenate lists
11. Exit

Enter your choice: 11

Process returned 0 (0x0) execution time : 1675.154 s

Press any key to continue.

```

Start here x singlylist.c x SLL.c x s.c x sqsll.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  // -----
5  // Node Structure
6  // -----
7  struct Node {
8      int data;
9      struct Node* next;
10 };
11
12 // Create new node
13 struct Node* createNode(int value) {
14     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
15     newNode->data = value;
16     newNode->next = NULL;
17     return newNode;
18 }
19
20 // -----
21 // Stack Operations (using SLL)
22 // -----
23 void push(struct Node** top, int value) {
24     struct Node* newNode = createNode(value);
25     newNode->next = *top;
26     *top = newNode;
27     printf("Pushed: %d\n", value);
28 }
29
30 void pop(struct Node** top) {
31     if (*top == NULL) {
32         printf("Stack Underflow!\n");
33         return;
34     }
35     struct Node* temp = *top;
36     printf("Popped: %d\n", temp->data);
37     *top = (*top)->next;
38     free(temp);
39 }
40

```

```
Start here x singlylist.c x SLL.c x s.c x sqsll.c x
40
41 void displayStack(struct Node* top) {
42     if (top == NULL) {
43         printf("Stack is Empty\n");
44         return;
45     }
46     printf("Stack (Top to Bottom): ");
47     while (top != NULL) {
48         printf("%d ", top->data);
49         top = top->next;
50     }
51     printf("\n");
52 }
53
54 // -----
55 // Queue Operations (using SLL)
56 // -----
57 void enqueue(struct Node** front, struct Node** rear, int value) {
58     struct Node* newNode = createNode(value);
59
60     if (*rear == NULL) { // First element
61         *front = *rear = newNode;
62     } else {
63         (*rear)->next = newNode;
64         *rear = newNode;
65     }
66     printf("Enqueued: %d\n", value);
67 }
68
69 void dequeue(struct Node** front, struct Node** rear) {
70     if (*front == NULL) {
71         printf("Queue Underflow!\n");
72         return;
73     }
74
75     struct Node* temp = *front;
76     printf("Dequeued: %d\n", temp->data);
77     *front = (*front)->next;
78
79     if (*front == NULL) // Queue becomes empty
80         *rear = NULL;
81
82     free(temp);
83 }
84
```

```

85 void displayQueue(struct Node* front) {
86     if (front == NULL) {
87         printf("Queue is Empty\n");
88         return;
89     }
90     printf("Queue (Front to Rear): ");
91     while (front != NULL) {
92         printf("%d ", front->data);
93         front = front->next;
94     }
95     printf("\n");
96 }
97
98 // -----
99 // Main Program (Menu Driven)
100 // -----
101 int main() {
102     struct Node* stackTop = NULL;
103     struct Node* qFront = NULL;
104     struct Node* qRear = NULL;
105
106     int choice, value;
107
108     while (1) {
109         printf("\n--- MENU ---\n");
110         printf("1. Push (Stack)\n");
111         printf("2. Pop (Stack)\n");
112         printf("3. Display Stack\n");
113         printf("4. Enqueue (Queue)\n");
114         printf("5. Dequeue (Queue)\n");
115         printf("6. Display Queue\n");
116         printf("7. Exit\n");
117         printf("Enter your choice: ");
118         scanf("%d", &choice);
119

```

```
120
121
122     switch (choice) {
123     case 1:
124         printf("Enter value to push: ");
125         scanf("%d", &value);
126         push(&stackTop, value);
127         break;
128
129     case 2:
130         pop(&stackTop);
131         break;
132
133     case 3:
134         displayStack(stackTop);
135         break;
136
137     case 4:
138         printf("Enter value to enqueue: ");
139         scanf("%d", &value);
140         enqueue(&qFront, &qRear, value);
141         break;
142
143     case 5:
144         dequeue(&qFront, &qRear);
145         break;
146
147     case 6:
148         displayQueue(qFront);
149         break;
150
151     case 7:
152         printf("Exiting...\n");
153         exit(0);
154
155     default:
156         printf("Invalid Choice!\n");
157     }
158 }
159 return 0;
160 }
```



"C:\Users\BMSCE\Documents" X



--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 1

Enter value to push: 10

Pushed: 10

--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 1

Enter value to push: 20

Pushed: 20

--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 1

Enter value to push: 30

Pushed: 30

--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 2

Popped: 30



"C:\Users\BMSCE\Documents" X



--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 3

Stack (Top to Bottom): 20 10

--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 4

Enter value to enqueue: 100

Enqueued: 100

--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 4

Enter value to enqueue: 200

Enqueued: 200

--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 4

Enter value to enqueue: 300

Enqueued: 300

--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 5

Dequeued: 100

--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 6

Queue (Front to Rear): 200 300

--- MENU ---

1. Push (Stack)
2. Pop (Stack)
3. Display Stack
4. Enqueue (Queue)
5. Dequeue (Queue)
6. Display Queue
7. Exit

Enter your choice: 7

Exiting...

Process returned 0 (0x0) execution time : 29.929 s

Press any key to continue.

|