WEEK-07

Start here | singlylist.c | SLL.c

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

// Define a node structure
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new element at the beginning
void insertAtFirst(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

// Function to insert a new element at the end
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

```

```c

// Function to insert a new element at a specific position
void insertAtPosition(struct Node** head, int data, int position) {
    if (position == 0) {
        insertAtFirst(head, data);
        return;
    }
    struct Node* newNode = createNode(data);
    struct Node* temp = *head;
    for (int i = 0; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Position out of range\n");
        free(newNode);
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}

// Function to delete a node from the beginning
void deleteAtFirst(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty, nothing to delete.\n");
        return;
    }
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
    printf("Node deleted from the beginning.\n");
}

```

Start here | singlylist.c | SLL.c

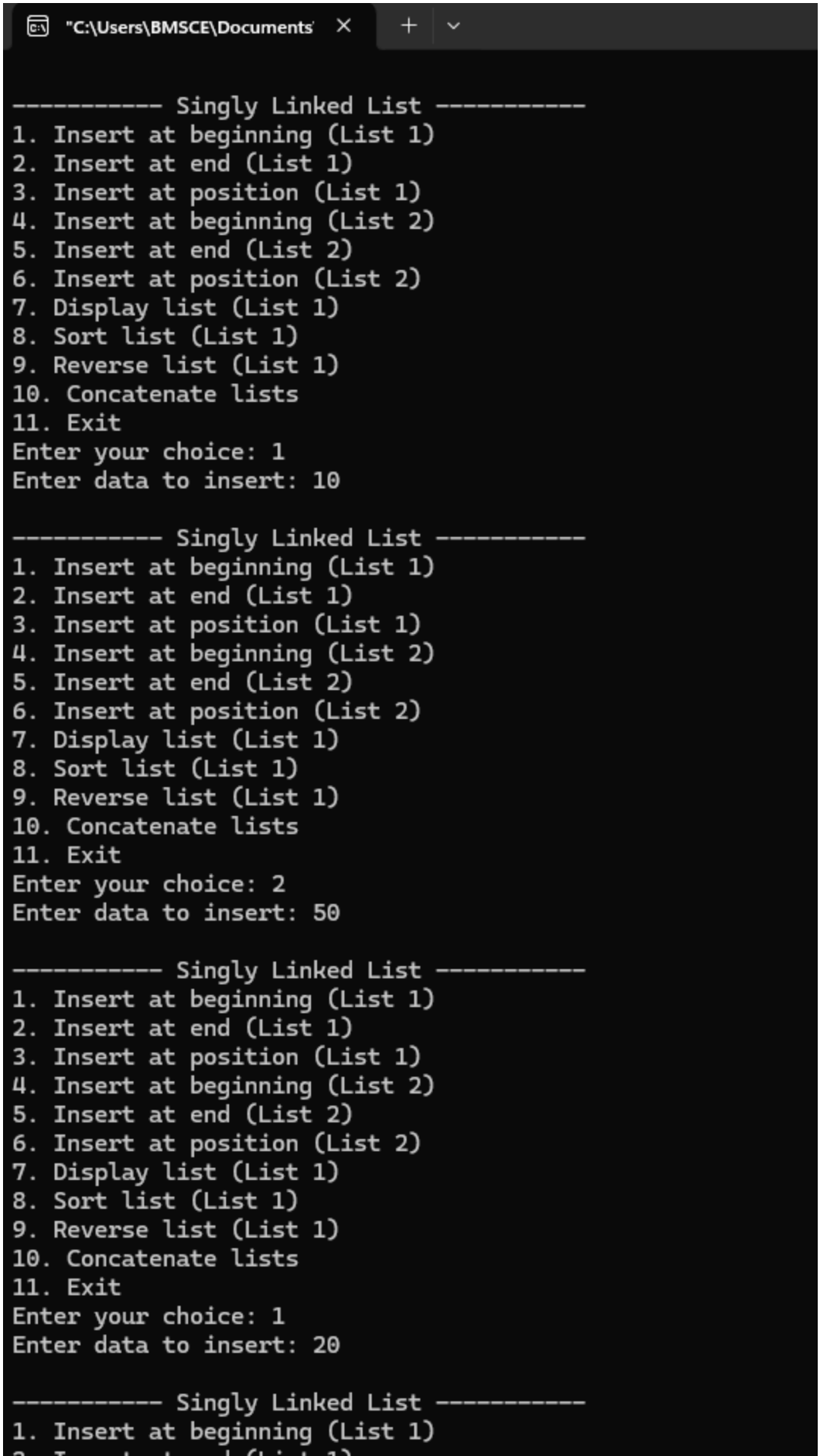
```c
void deleteAtEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty, nothing to delete.\n");
        return;
    }
    struct Node* temp = *head;
    if (temp->next == NULL) { // Only one node in the list
        free(temp);
        *head = NULL;
        printf("Node deleted from the end.\n");
        return;
    }
    while (temp->next != NULL && temp->next->next != NULL) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
    printf("Node deleted from the end.\n");
}

// Function to delete a node from a specific position
void deleteAtPosition(struct Node** head, int position) {
    if (*head == NULL) {
        printf("List is empty, nothing to delete.\n");
        return;
    }
    if (position == 0) {
        deleteAtFirst(head);
        return;
    }
    struct Node* temp = *head;
    for (int i = 0; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("Position out of range\n");
        return;
    }
    struct Node* nodeToDelete = temp->next;
    temp->next = temp->next->next;
    free(nodeToDelete);
    printf("Node deleted from position %d.\n", position);
}
```

```c
// Function to print the linked list
void print(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to sort the linked list (Bubble Sort)
void sortList(struct Node** head) {
    if (*head == NULL) {
        return;
    }
    struct Node *i, *j;
    int temp;
    for (i = *head; i != NULL; i = i->next) {
        for (j = i->next; j != NULL; j = j->next) {
            if (i->data > j->data) {
                temp = i->data;
                i->data = j->data;
                j->data = temp;
            }
        }
    }
    printf("List sorted.\n");
}

// Function to reverse the linked list
void reverseList(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head = prev;
    printf("List reversed.\n");
}

// Function to concatenate two linked lists
void concatenateLists(struct Node** head1, struct Node* head2) {
    if (*head1 == NULL) {
        *head1 = head2;
        return;
    }
    struct Node* temp = *head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
    printf("Lists concatenated.\n");
}
```

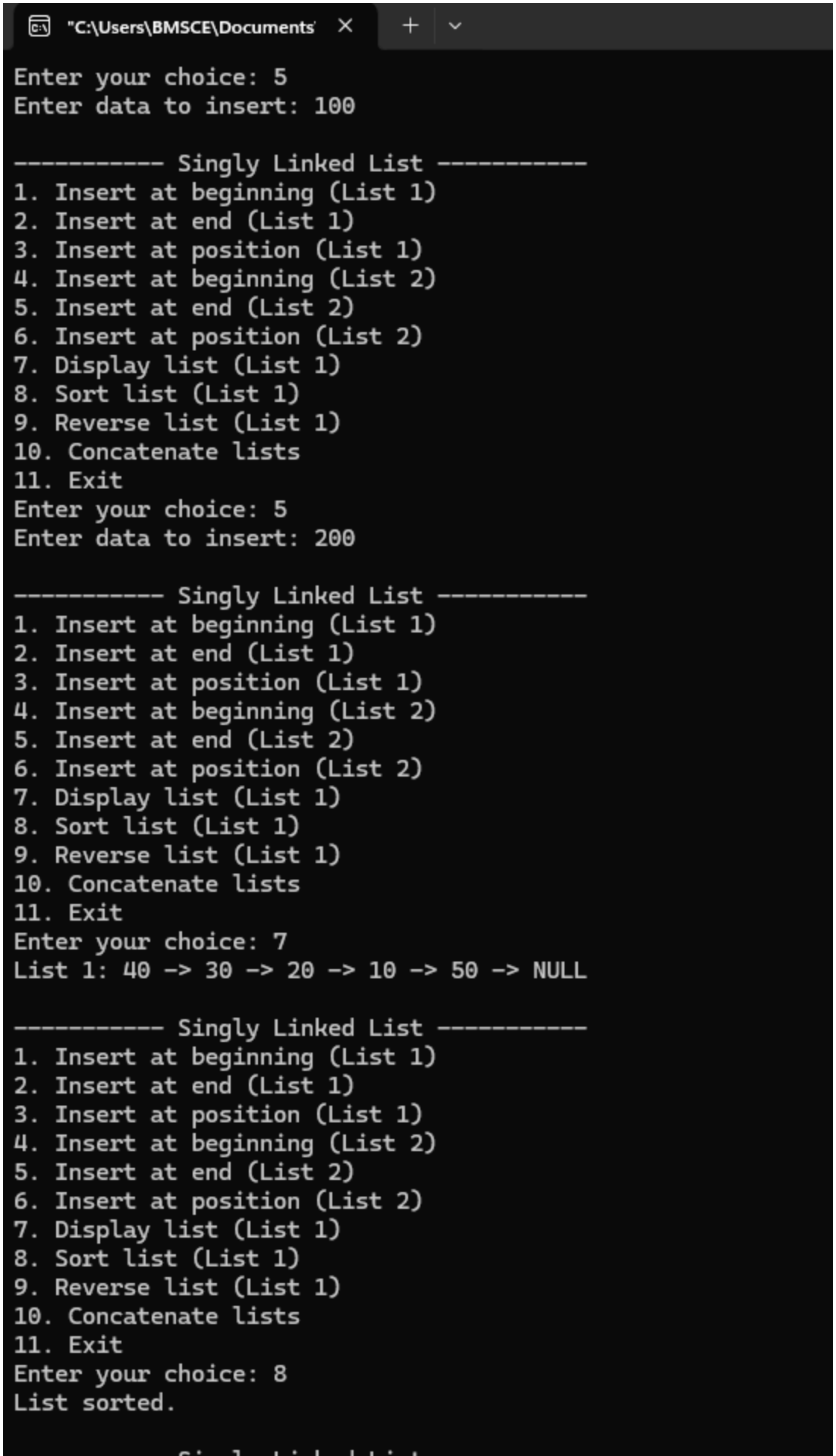
Start here | singlylist.c | SLL.c

```c
// Main function
int main() {
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    int choice, data, position;

    while (1) {
        printf("\n---------- Singly Linked List ----------\n");
        printf("1. Insert at beginning (List 1)\n");
        printf("2. Insert at end (List 1)\n");
        printf("3. Insert at position (List 1)\n");
        printf("4. Insert at beginning (List 2)\n");
        printf("5. Insert at end (List 2)\n");
        printf("6. Insert at position (List 2)\n");
        printf("7. Display list (List 1)\n");
        printf("8. Sort list (List 1)\n");
        printf("9. Reverse list (List 1)\n");
        printf("10. Concatenate lists\n");
        printf("11. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
        case 1:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            insertAtFirst(&head1, data);
            break;
        case 2:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            insertAtEnd(&head1, data);
            break;
        case 3:
            printf("Enter data to insert: ");
            scanf("%d", &data);
            printf("Enter position (starting from 0): ");
            scanf("%d", &position);
            insertAtPosition(&head1, data, position);
            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
```


"C:\Users\BMSCE\Documents
---------- 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)

```
"C:\Users\BMSCE\Documents
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 ----------
```


"C:\Users\BMSCE\Documents
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.

```
"C:\Users\BMSCE\Documents
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 | singlylist.c | SLL.c | s.c | sqsll.c

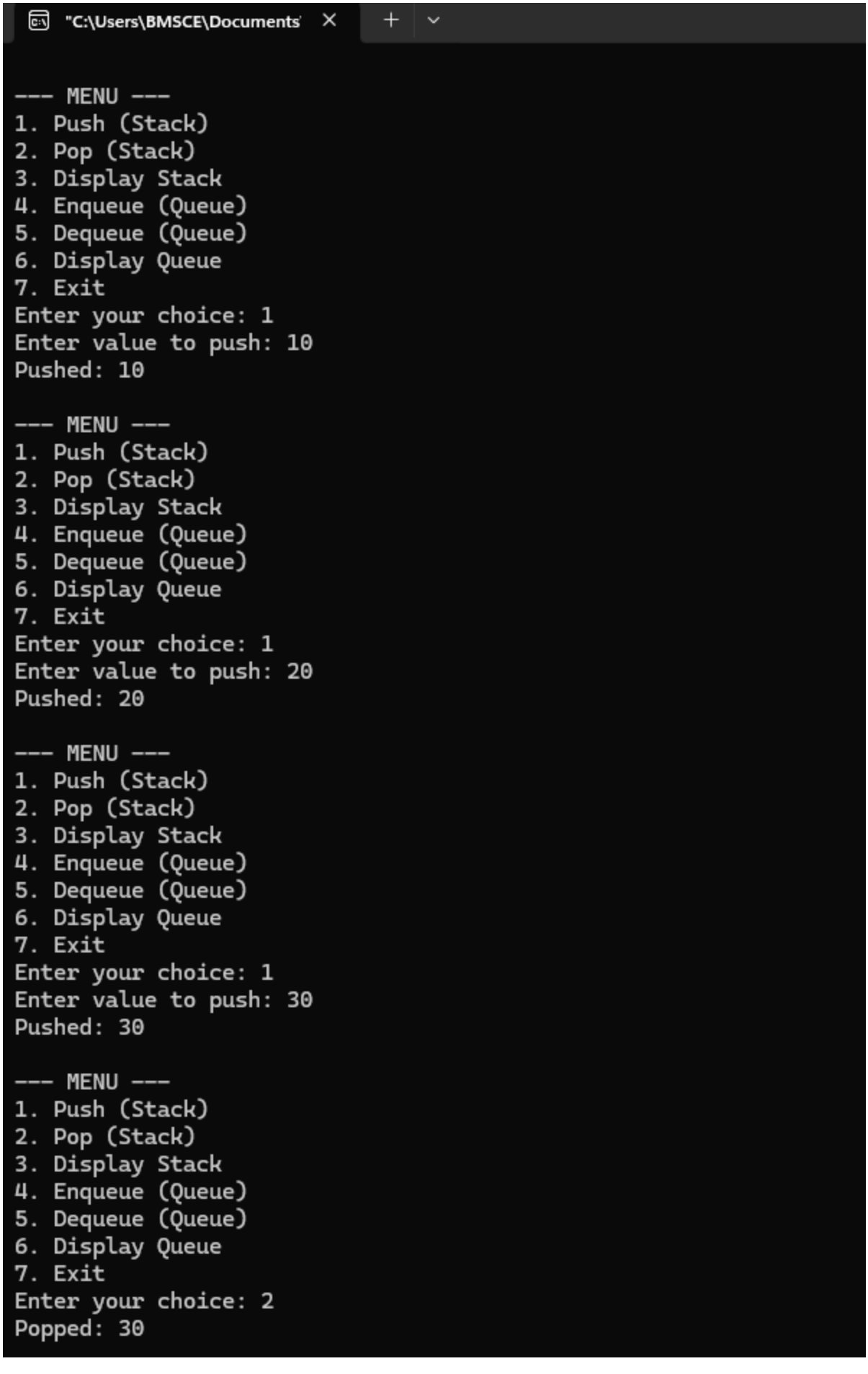
```c
#include <stdio.h>
#include <stdlib.h>

// --------------------------------
// Node Structure
// --------------------------------
struct Node {
    int data;
    struct Node* next;
};

// Create new node
struct Node* createNode(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

// --------------------------------
// Stack Operations (using SLL)
// --------------------------------
void push(struct Node** top, int value) {
    struct Node* newNode = createNode(value);
    newNode->next = *top;
    *top = newNode;
    printf("Pushed: %d\n", value);
}

void pop(struct Node** top) {
    if (*top == NULL) {
        printf("Stack Underflow!\n");
        return;
    }
    struct Node* temp = *top;
    printf("Popped: %d\n", temp->data);
    *top = (*top)->next;
    free(temp);
}
```

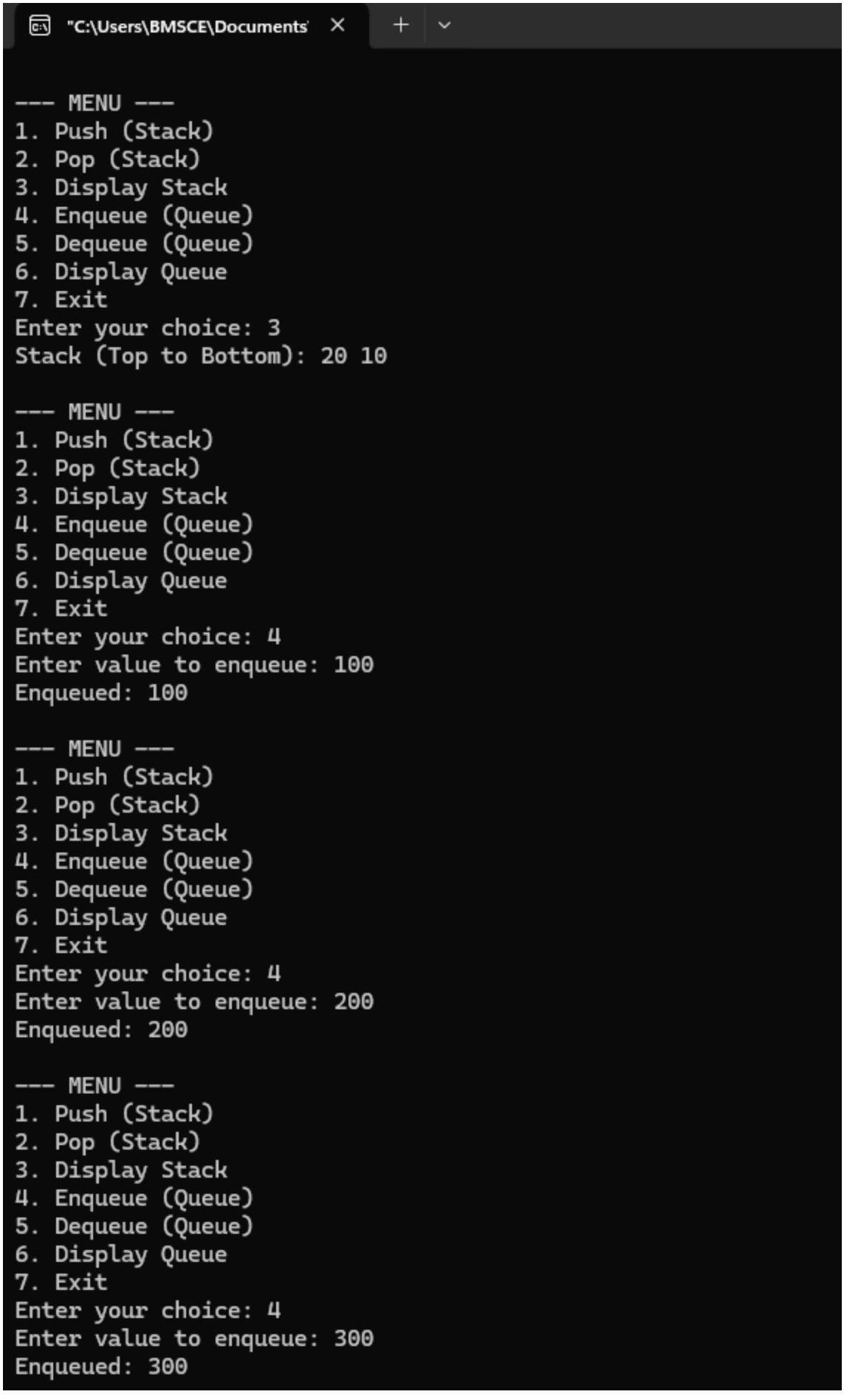
Start here | singlylist.c | SLL.c | s.c | sqsll.c

```c

void displayStack(struct Node* top) {
    if (top == NULL) {
        printf("Stack is Empty\n");
        return;
    }
    printf("Stack (Top to Bottom): ");
    while (top != NULL) {
        printf("%d ", top->data);
        top = top->next;
    }
    printf("\n");
}

// --------------------------------
// Queue Operations (using SLL)
// --------------------------------
void enqueue(struct Node** front, struct Node** rear, int value) {
    struct Node* newNode = createNode(value);

    if (*rear == NULL) {       // First element
        *front = *rear = newNode;
    } else {
        (*rear)->next = newNode;
        *rear = newNode;
    }
    printf("Enqueued: %d\n", value);
}

void dequeue(struct Node** front, struct Node** rear) {
    if (*front == NULL) {
        printf("Queue Underflow!\n");
        return;
    }

    struct Node* temp = *front;
    printf("Dequeued: %d\n", temp->data);
    *front = (*front)->next;

    if (*front == NULL)  // Queue becomes empty
        *rear = NULL;

    free(temp);
}

```

```c
void displayQueue(struct Node* front) {
    if (front == NULL) {
        printf("Queue is Empty\n");
        return;
    }
    printf("Queue (Front to Rear): ");
    while (front != NULL) {
        printf("%d ", front->data);
        front = front->next;
    }
    printf("\n");
}

// -------------------------------
// Main Program (Menu Driven)
// -------------------------------
int main() {
    struct Node* stackTop = NULL;
    struct Node* qFront = NULL;
    struct Node* qRear = NULL;

    int choice, value;

    while (1) {
        printf("\n--- MENU ---\n");
        printf("1. Push (Stack)\n");
        printf("2. Pop (Stack)\n");
        printf("3. Display Stack\n");
        printf("4. Enqueue (Queue)\n");
        printf("5. Dequeue (Queue)\n");
        printf("6. Display Queue\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

```

```c
        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&stackTop, value);
                break;

            case 2:
                pop(&stackTop);
                break;

            case 3:
                displayStack(stackTop);
                break;

            case 4:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(&qFront, &qRear, value);
                break;

            case 5:
                dequeue(&qFront, &qRear);
                break;

            case 6:
                displayQueue(qFront);
                break;

            case 7:
                printf("Exiting...\n");
                exit(0);

            default:
                printf("Invalid Choice!\n");
        }
    }
    return 0;
}
```


"C:\Users\BMSCE\Documents
--- 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
--- 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.
```