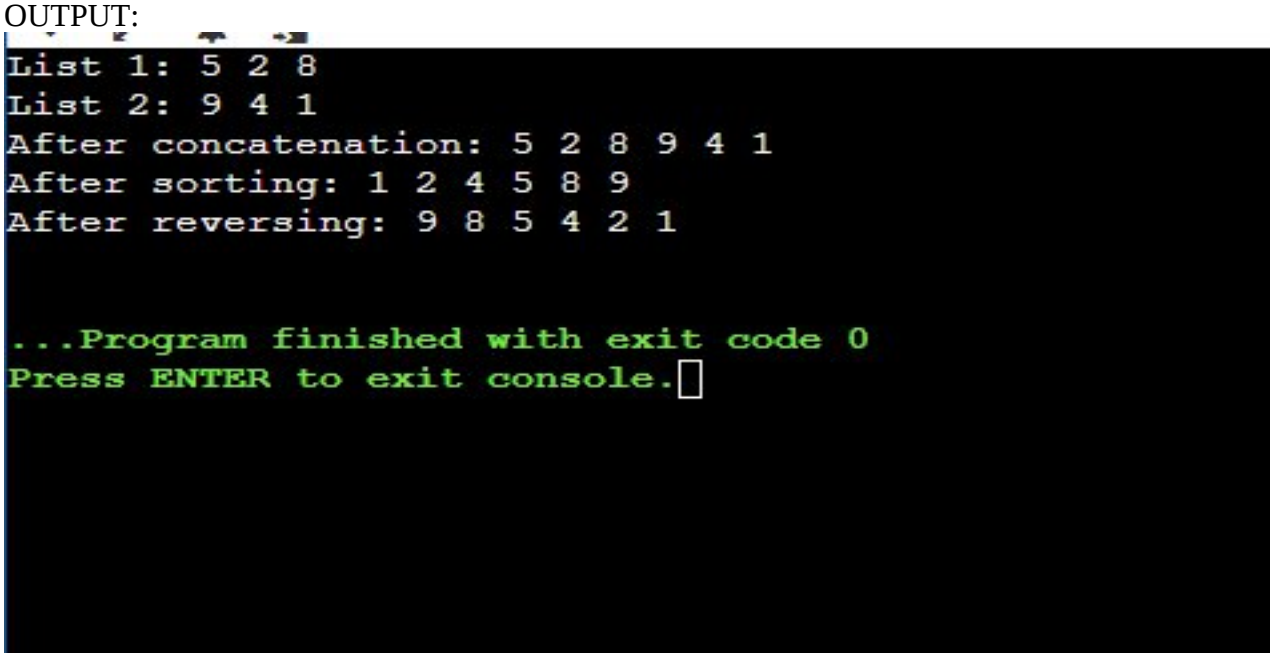
```
#include <stdio.h>
#include <stdlib.h>

// 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 == NULL) {
        printf("Memory allocation failed!\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the list
void insertEnd(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;
}

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

// Function to reverse the linked list
void reverse(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* nextNode = NULL;
    while (current != NULL) {
        nextNode = current->next;
```

```
        current->next = prev;
        prev = current;
        current = nextNode;
    }
    *head = prev;
}

// Function to concatenate two linked lists
void concatenate(struct Node** list1, struct Node* list2) {
    if (*list1 == NULL) {
        *list1 = list2;
        return;
    }
    struct Node* temp = *list1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = list2;
}

// Function to sort the linked list (bubble sort)
void sort(struct Node** head) {
    struct Node *current, *nextNode;
    int temp;
    current = *head;
    if (current == NULL) {
        printf("List is empty\n");
        return;
    }
    while (current->next != NULL) {
        nextNode = current->next;
        while (nextNode != NULL) {
            if (current->data > nextNode->data) {
                temp = current->data;
                current->data = nextNode->data;
                nextNode->data = temp;
            }
            nextNode = nextNode->next;
        }
        current = current->next;
    }
}

int main() {
    struct Node* list1 = NULL;
    struct Node* list2 = NULL;

    // Insert elements into list1
    insertEnd(&list1, 5);
    insertEnd(&list1, 2);
    insertEnd(&list1, 8);
    printf("List 1: ");
```

```
    display(list1);

    // Insert elements into list2
    insertEnd(&list2, 9);
    insertEnd(&list2, 4);
    insertEnd(&list2, 1);
    printf("List 2: ");
    display(list2);

    // Concatenate list2 to list1
    concatenate(&list1, list2);
    printf("After concatenation: ");
    display(list1);

    // Sort the concatenated list
    sort(&list1);
    printf("After sorting: ");
    display(list1);

    // Reverse the sorted list
    reverse(&list1);
    printf("After reversing: ");
    display(list1);

    return 0;
}
```

OUTPUT:



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

// Node structure for doubly linked list
struct Node {
    int data;
    struct Node* prev;
    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 == NULL) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new node to the left of a given node
void insertLeft(struct Node** head, struct Node* givenNode, int newData) {
    struct Node* newNode = createNode(newData);
    if (givenNode == NULL) {
        printf("Given node cannot be NULL\n");
        return;
    }
    newNode->next = givenNode;
    newNode->prev = givenNode->prev;
    if (givenNode->prev != NULL) {
        givenNode->prev->next = newNode;
    }
    givenNode->prev = newNode;
    if (*head == givenNode) {
        *head = newNode;
    }
}

// Function to delete the node based on a specific value
void deleteNode(struct Node** head, int key) {
    struct Node* temp = *head;
    while (temp != NULL && temp->data != key) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Node with value %d not found\n", key);
        return;
    }
    if (temp->prev != NULL) {
```

```
        temp->prev->next = temp->next;
    }
    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }
    if (temp == *head) {
        *head = temp->next;
    }
    free(temp);
}

// Function to display the contents of the list
void displayList(struct Node* head) {
    printf("Doubly linked list: ");
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

// Main function
int main() {
    struct Node* head = NULL;

    // Creating the doubly linked list
    head = createNode(1);
    head->next = createNode(2);
    head->next->prev = head;
    head->next->next = createNode(3);
    head->next->next->prev = head->next;

    // Displaying the initial list
    displayList(head);

    // Inserting a new node to the left of a given node
    insertLeft(&head, head->next, 5);
    displayList(head);

    // Deleting a node based on a specific value
    deleteNode(&head, 2);
    displayList(head);

    return 0;
}
```

OUTPUT:

```
Doubly linked list: 1 2 3
Doubly linked list: 1 5 2 3
Doubly linked list: 1 5 3


...Program finished with exit code 0
Press ENTER to exit console.
```