```
=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 2
Enter value to insert into List2: 5

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 2
Enter value to insert into List2: 8

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 3
List1: 2 -> 4 -> 5 -> 8 -> NULL

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 4
```

```
=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 4
List2: 9 -> 7 -> 5 -> 8 -> NULL

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 5
List sorted.

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 6
List reversed.

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 7
Lists concatenated.
```

```
=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 1
Enter value to insert into List1: 8

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 2
Enter value to insert into List2: 9

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 2
Enter value to insert into List2: 7

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 2
```

```c
int main() {
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    struct Node* result = NULL;
    int choice, value;

    while (1) {
        printf("\n--- Linked List Operations ---\n");
        printf("1. Insert into List1\n");
        printf("2. Insert into List2\n");
        printf("3. Display List1\n");
        printf("4. Display List2\n");
        printf("5. Sort List1\n");
        printf("6. Reverse List1\n");
        printf("7. Concatenate List1 & List2 -> Result\n");
        printf("8. Display Result List\n");
        printf("9. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert into List1: ");
                scanf("%d", &value);
                insertEnd(&head1, value);
                break;
            case 2:
                printf("Enter value to insert into List2: ");
                scanf("%d", &value);
                insertEnd(&head2, value);
                break;
            case 3:
                printf("List1: ");
                displayList(head1);
                break;
            case 4:
                printf("List2: ");
                displayList(head2);
                break;
            case 5:
                sortList(head1);
                break;
            case 6:
                reverseList(&head1);
                break;
            case 7:
                result = concatenateLists(head1, head2);
                printf("Lists concatenated.\n");
```

```
        printf("8. Display Result List\n");
        printf("9. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert into List1: ");
                scanf("%d", &value);
                insertEnd(&head1, value);
                break;
            case 2:
                printf("Enter value to insert into List2: ");
                scanf("%d", &value);
                insertEnd(&head2, value);
                break;
            case 3:
                printf("List1: ");
                displayList(head1);
                break;
            case 4:
                printf("List2: ");
                displayList(head2);
                break;
            case 5:
                sortList(head1);
                break;
            case 6:
                reverseList(&head1);
                break;
            case 7:
                result = concatenateLists(head1, head2);
                printf("Lists concatenated.\n");
                break;
            case 8:
                printf("Result List: ");
                displayList(result);
                break;
            case 9:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice, try again.\n");
        }
    }
    return 0;
}
```

there are changes to them. Save the configuration | C/C++ | Windows (CR+LF) | WINDOWS-1252 | Line 97, Col 24, Pos 2273 | Insert | Read/Write | default

Search

ENG IN 11:59:52 17-11-2025

```
=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 1
Enter value to insert into List1: 2

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 1
Enter value to insert into List1: 4

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 1
Enter value to insert into List1: 5

=== Linked List Operations ===
1. Insert into List1
2. Insert into List2
3. Display List1
4. Display List2
5. Sort List1
6. Reverse List1
7. Concatenate List1 & List2 -> Result
8. Display Result List
9. Exit
Enter choice: 1
```

```c
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");
}

void sortList(struct Node* head) {
    if (head == NULL || head->next == NULL) {
        printf("List too short to sort.\n");
        return;
    }
    struct Node* i;
    struct Node* 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");
}


struct Node* concatenateLists(struct Node* head1, struct Node* head2) {
    if (head1 == NULL) return head2;
    struct Node* temp = head1;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = head2;
    return head1;
}

int main() {
    struct Node* head1 = NULL;
    struct Node* head2 = NULL;
    struct Node* result = NULL;
```

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

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

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

void insertEnd(struct Node** head, int value) {
    struct Node* newNode = createNode(value);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}

void displayList(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");
}

void reverseList(struct Node** head) {
    struct Node* prev = NULL;
    struct Node* current = *head;
    struct Node* next = NULL;
    while (current != NULL) {
        next = current->next;
```