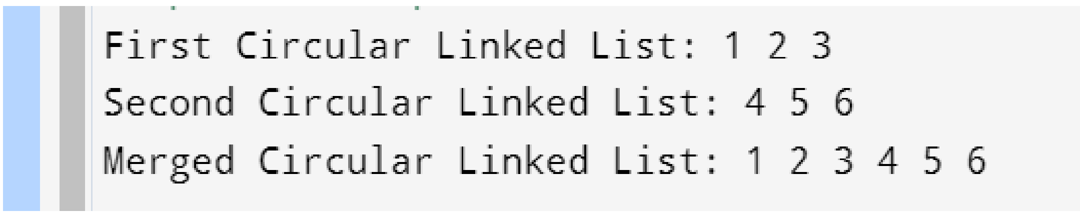
Tabish Ahmad

Assignment 9 –Advanced Linked List

23103145

Question 1)

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;};
void insert(Node** head, int data) {
   Node* newNode = new Node();
   newNode->data = data;
   if (*head == nullptr) {
      *head = newNode;
      newNode->next = *head;
   } else {
      Node* temp = *head;
      while (temp->next != *head) {
         temp = temp->next;}
      temp->next = newNode;
      newNode->next = *head;}}
Node* mergeCircularLists(Node* head1, Node* head2) {
   if (head1 == nullptr) return head2;
   if (head2 == nullptr) return head1;
   Node* temp1 = head1;
   while (temp1->next != head1) {
      temp1 = temp1->next;}
   Node* temp2 = head2;
   while (temp2->next != head2) {
      temp2 = temp2->next;}
   temp1->next = head2;
   temp2->next = head1;
   return head1;}
void display(Node* head) {
   if (head == nullptr) {
      cout << "List is empty." << endl;
      return;}
   Node* temp = head;
   do {
      cout << temp->data << " ";
      temp = temp->next;
   } while (temp != head);
   cout << endl;}
int main() {
   Node* head1 = nullptr;
   Node* head2 = nullptr;
   insert(&head1, 1);
   insert(&head1, 2);
   insert(&head1, 3);
   insert(&head2, 4);
   insert(&head2, 5);
   insert(&head2, 6);
   cout << "First Circular Linked List: ";
   display(head1);
```

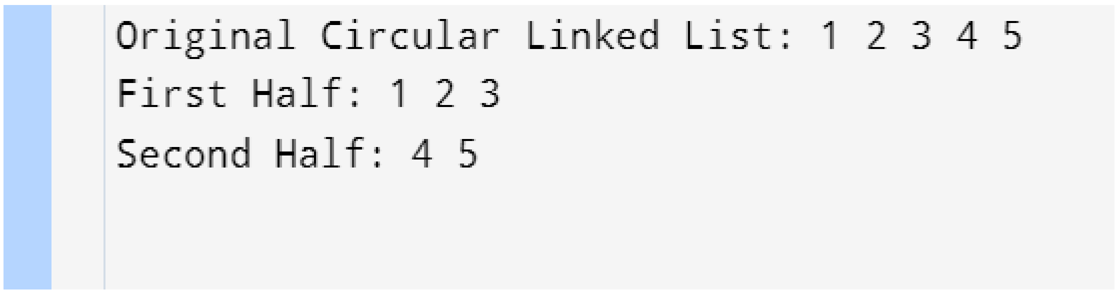


```
First Circular Linked List: 1 2 3
Second Circular Linked List: 4 5 6
Merged Circular Linked List: 1 2 3 4 5 6
```

```
    cout << "Second Circular Linked List: ";
    display(head2);
    Node* mergedHead = mergeCircularLists(head1, head2);
    cout << "Merged Circular Linked List: ";
    display(mergedHead);
    return 0;}
```

Question 2)

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;};
void insert(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (*head == nullptr) {
        *head = newNode;
        newNode->next = *head;
    } else {
        Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = *head;}}
void splitCircularList(Node* head, Node** head1, Node** head2) {
    if (head == nullptr || head->next == head) {
        *head1 = head;
        *head2 = nullptr;
        return;}
    Node* slow = head;
    Node* fast = head;
    while (fast->next != head && fast->next->next != head) {
        slow = slow->next;
        fast = fast->next->next;}
    if (fast->next->next == head) {
        fast = fast->next;}
    *head1 = head;
    *head2 = slow->next;
    slow->next = *head1;
    fast->next = *head2;}
void display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;}
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;}
int main() {
    Node* head = nullptr;
    Node* head1 = nullptr;
    Node* head2 = nullptr;
    insert(&head, 1);
```

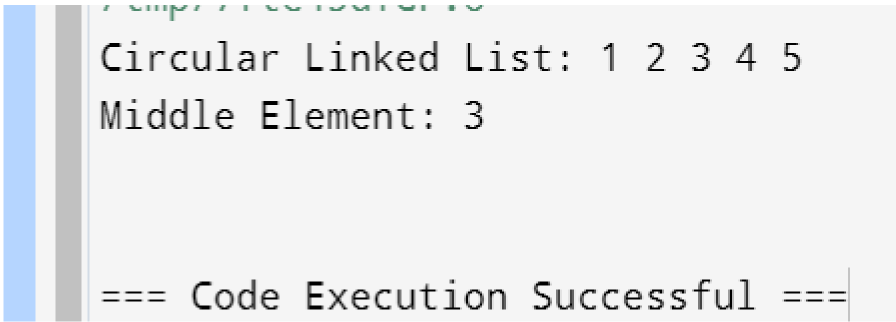


```
Original Circular Linked List: 1 2 3 4 5
First Half: 1 2 3
Second Half: 4 5
```

```
    insert(&head, 2);
    insert(&head, 3);
    insert(&head, 4);
    insert(&head, 5);
    cout << "Original Circular Linked List: ";
    display(head);
    splitCircularList(head, &head1, &head2);
    cout << "First Half: ";
    display(head1);
    cout << "Second Half: ";
    display(head2);
    return 0;}
```

Question 3)

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;};
void insert(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (*head == nullptr) {
        *head = newNode;
        newNode->next = *head;
    } else {
        Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;}
        temp->next = newNode;
        newNode->next = *head;}}
int findMiddle(Node* head) {
    if (head == nullptr) {
        return -1;
    }
    Node* slow = head;
    Node* fast = head;
    while (fast->next != head && fast->next->next != head) {
        slow = slow->next;
        fast = fast->next->next;}
    return slow->data;}
void display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;}
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;}
int main() {
    Node* head = nullptr;
    insert(&head, 1);
    insert(&head, 2);
    insert(&head, 3);
    insert(&head, 4);
    insert(&head, 5);
```



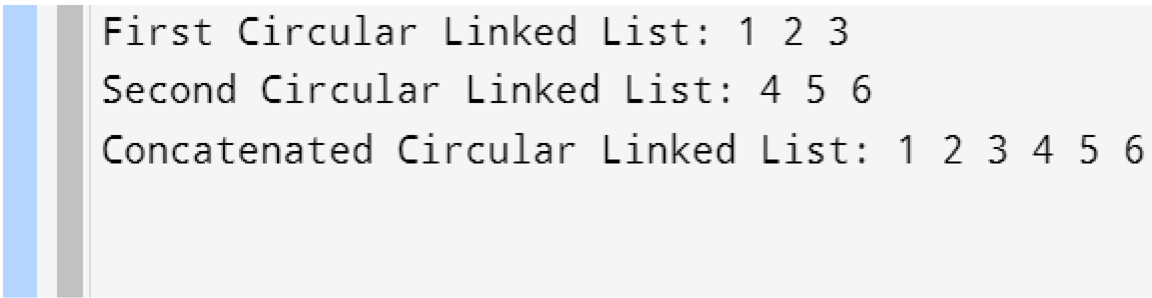
```
Circular Linked List: 1 2 3 4 5
Middle Element: 3


=== Code Execution Successful ===
  cout << "Circular Linked List: ";
  display(head);
  int middle = findMiddle(head);
  if (middle != -1) {
    cout << "Middle Element: " << middle << endl;
  } else {
    cout << "The list is empty." << endl;}
  return 0;}
```

Question 4)

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  Node* next;};
void insert(Node** head, int data) {
  Node* newNode = new Node();
  newNode->data = data;
  if (*head == nullptr) {
    *head = newNode;
    newNode->next = *head;
  } else {
    Node* temp = *head;
    while (temp->next != *head) {
      temp = temp->next;}
    temp->next = newNode;
    newNode->next = *head;}}
Node* concatenate(Node* head1, Node* head2) {
  if (head1 == nullptr) return head2;
  if (head2 == nullptr) return head1;

  Node* temp1 = head1;
  while (temp1->next != head1) {
    temp1 = temp1->next;}
  Node* temp2 = head2;
  while (temp2->next != head2) {
    temp2 = temp2->next;}
  temp1->next = head2;
  temp2->next = head1;
  return head1;}
void display(Node* head) {
  if (head == nullptr) {
    cout << "List is empty." << endl;
    return;}
  Node* temp = head;
  do {
    cout << temp->data << " ";
    temp = temp->next;
  } while (temp != head);
  cout << endl;}
int main() {
  Node* head1 = nullptr;
  Node* head2 = nullptr;
  insert(&head1, 1);
  insert(&head1, 2);
  insert(&head1, 3);
  insert(&head2, 4);
  insert(&head2, 5);
```

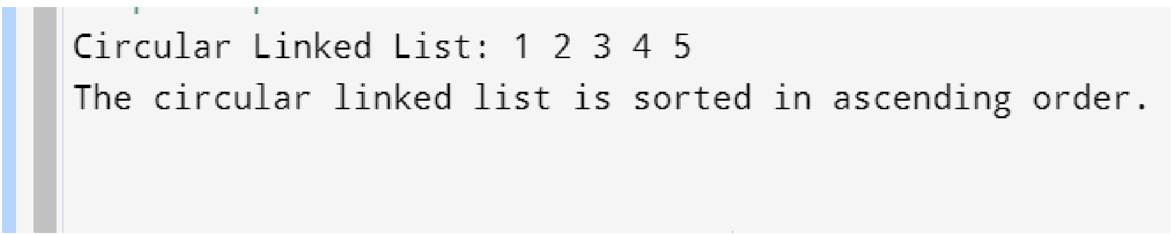


```
First Circular Linked List: 1 2 3
Second Circular Linked List: 4 5 6
Concatenated Circular Linked List: 1 2 3 4 5 6
```

```
    insert(&head2, 6);
    cout << "First Circular Linked List: ";
    display(head1);
    cout << "Second Circular Linked List: ";
    display(head2);
    Node* concatenatedHead = concatenate(head1, head2);
    cout << "Concatenated Circular Linked List: ";
    display(concatenatedHead);
    return 0;}
```

Question 5)

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;};
void insert(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    if (*head == nullptr) {
        *head = newNode;
        newNode->next = *head;
    } else {
        Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;}
        temp->next = newNode;
        newNode->next = *head;}}
bool isSorted(Node* head) {
    if (head == nullptr || head->next == head) {
        return true;}
    Node* temp = head;
    do {
        if (temp->data > temp->next->data) {
            return false;}
        temp = temp->next;
    } while (temp->next != head);
    return true;}
void display(Node* head) {
    if (head == nullptr) {
        cout << "List is empty." << endl;
        return;}
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;}
int main() {
    Node* head = nullptr;
    insert(&head, 1);
    insert(&head, 2);
    insert(&head, 3);
    insert(&head, 4);
    insert(&head, 5);
    cout << "Circular Linked List: ";
    display(head);
```

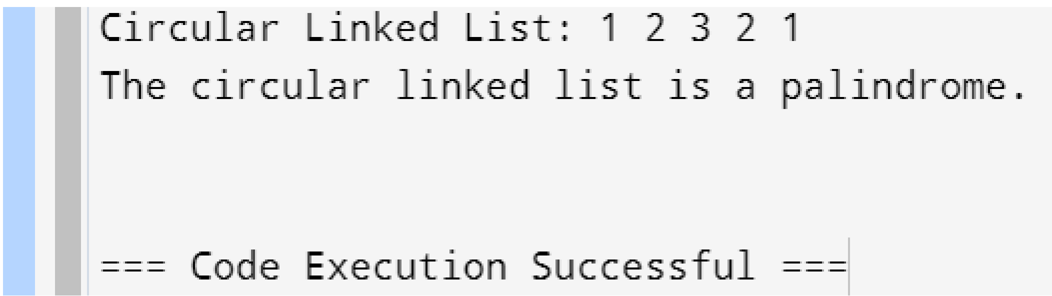


```
Circular Linked List: 1 2 3 4 5
The circular linked list is sorted in ascending order.
```

```
  if (isSorted(head)) {
    cout << "The circular linked list is sorted in ascending order." << endl;
  } else {
    cout << "The circular linked list is not sorted in ascending order." << endl;}
  return 0;}
```

Question 6)

```
#include <iostream>
#include <stack>
using namespace std;
struct Node {
  int data;
  Node* next;};
void insert(Node** head, int data) {
  Node* newNode = new Node();
  newNode->data = data;
  if (*head == nullptr) {
    *head = newNode;
    newNode->next = *head;
  } else {
    Node* temp = *head;
    while (temp->next != *head) {
      temp = temp->next;}
    temp->next = newNode;
    newNode->next = *head;}}
bool isPalindrome(Node* head) {
  if (head == nullptr || head->next == head) {
    return true;}
  stack<int> s;
  Node* temp = head;
  do {
    s.push(temp->data);
    temp = temp->next;
  } while (temp != head);
  temp = head;
  do {
    int top = s.top();
    s.pop();
    if (temp->data != top) {
      return false;}
    temp = temp->next;
  } while (temp != head);
  return true;}
void display(Node* head) {
  if (head == nullptr) {
    cout << "List is empty." << endl;
    return;}
  Node* temp = head;
  do {
    cout << temp->data << " ";
    temp = temp->next;
  } while (temp != head);
  cout << endl;}
int main() {
  Node* head = nullptr;
  insert(&head, 1);
  insert(&head, 2);
  insert(&head, 3);
```



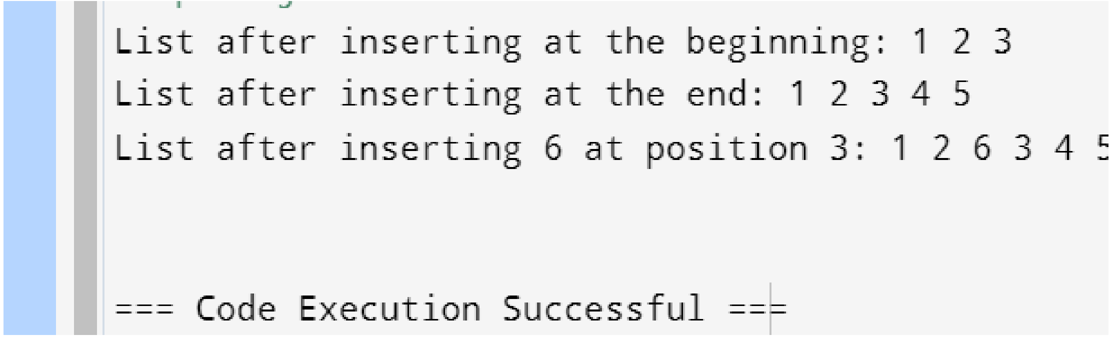
```
Circular Linked List: 1 2 3 2 1
The circular linked list is a palindrome.


=== Code Execution Successful ===
    insert(&head, 2);
    insert(&head, 1);
    cout << "Circular Linked List: ";
    display(head);
    if (isPalindrome(head)) {
        cout << "The circular linked list is a palindrome." << endl;
    } else {
        cout << "The circular linked list is not a palindrome." << endl;}
    return 0;}
```

Question 7)

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* prev;
    Node* next;};
void insertAtBeginning(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->prev = nullptr;
    newNode->next = *head;
    if (*head != nullptr) {
        (*head)->prev = newNode;}
    *head = newNode;}
void insertAtEnd(Node** head, int data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
    if (*head == nullptr) {
        newNode->prev = nullptr;
        *head = newNode;
        return;}
    Node* temp = *head;
    while (temp->next != nullptr) {
        temp = temp->next;}
    temp->next = newNode;
    newNode->prev = temp;}
void insertAtPosition(Node** head, int data, int position) {
    if (position <= 0) {
        cout << "Invalid position!" << endl;
        return;}
    Node* newNode = new Node();
    newNode->data = data;
    if (position == 1) {
        newNode->next = *head;
        newNode->prev = nullptr;
        if (*head != nullptr) {
            (*head)->prev = newNode;}
        *head = newNode;
        return;}
    Node* temp = *head;
    for (int i = 1; i < position - 1 && temp != nullptr; i++) {
        temp = temp->next;}
    if (temp == nullptr) {
        cout << "Position out of range!" << endl;
        delete newNode;
        return;}
```

```
  newNode->next = temp->next;
  newNode->prev = temp;
  if (temp->next != nullptr) {
    temp->next->prev = newNode;}
  temp->next = newNode;}
void display(Node* head) {
  if (head == nullptr) {
    cout << "List is empty." << endl;
    return;}
  Node* temp = head;
  while (temp != nullptr) {
    cout << temp->data << " ";
    temp = temp->next;}
  cout << endl;}
int main() {
  Node* head = nullptr;
  insertAtBeginning(&head, 3);
  insertAtBeginning(&head, 2);
  insertAtBeginning(&head, 1);
  cout << "List after inserting at the beginning: ";
  display(head);
  insertAtEnd(&head, 4);
  insertAtEnd(&head, 5);
  cout << "List after inserting at the end: ";
  display(head);
  insertAtPosition(&head, 6, 3);
  cout << "List after inserting 6 at position 3: ";
  display(head);
  return 0;}
```

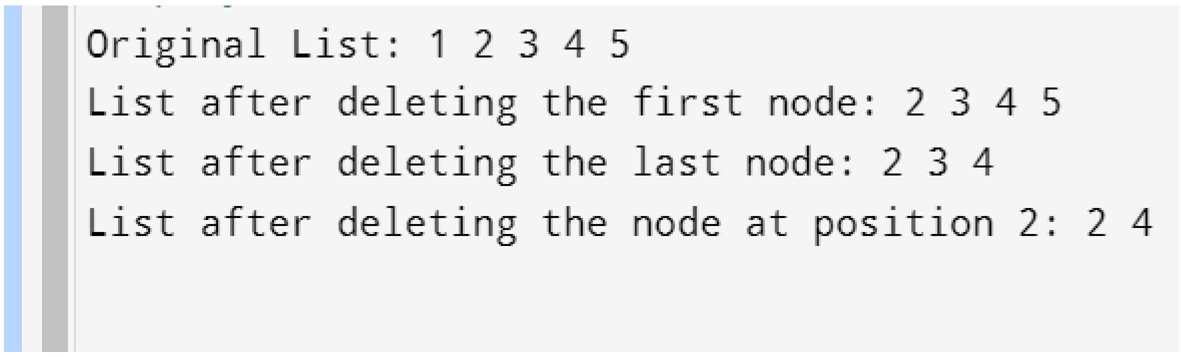


```
List after inserting at the beginning: 1 2 3
List after inserting at the end: 1 2 3 4 5
List after inserting 6 at position 3: 1 2 6 3 4 5


=== Code Execution Successful ===
```

Question 8)

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  Node* prev;
  Node* next;};
void insertAtEnd(Node** head, int data) {
  Node* newNode = new Node();
  newNode->data = data;
  newNode->next = nullptr;
  if (*head == nullptr) {
    newNode->prev = nullptr;
    *head = newNode;
    return;}
  Node* temp = *head;
  while (temp->next != nullptr) {
    temp = temp->next;}
  temp->next = newNode;
  newNode->prev = temp;}
void deleteFirstNode(Node** head) {
  if (*head == nullptr) {
    cout << "List is empty. No nodes to delete." << endl;
    return;}
  Node* temp = *head;
  *head = (*head)->next;
  if (*head != nullptr) {
    (*head)->prev = nullptr;}
  delete temp;}
void deleteLastNode(Node** head) {
  if (*head == nullptr) {
    cout << "List is empty. No nodes to delete." << endl;
    return;}
  Node* temp = *head;
  while (temp->next != nullptr) {
    temp = temp->next;}
  if (temp->prev != nullptr) {
    temp->prev->next = nullptr;
  } else {
    *head = nullptr;}
  delete temp;}
void deleteAtPosition(Node** head, int position) {
  if (*head == nullptr || position <= 0) {
    cout << "Invalid position or list is empty." << endl;
    return;}
  Node* temp = *head;
  for (int i = 1; temp != nullptr && i < position; i++) {
    temp = temp->next;}
  if (temp == nullptr) {
    cout << "Position out of range." << endl;
    return;}
  if (temp->prev != nullptr) {
    temp->prev->next = temp->next;
  } else {
    *head = temp->next;}
  if (temp->next != nullptr) {
    temp->next->prev = temp->prev;}
  delete temp;}
void display(Node* head) {
  if (head == nullptr) {
    cout << "List is empty." << endl;
    return;}
  Node* temp = head;
  while (temp != nullptr) {
    cout << temp->data << " ";
    temp = temp->next;}
  cout << endl;}
int main() {
  Node* head = nullptr;
  insertAtEnd(&head, 1);
  insertAtEnd(&head, 2);
  insertAtEnd(&head, 3);
  insertAtEnd(&head, 4);
  insertAtEnd(&head, 5);
  cout << "Original List: ";
  display(head);
  deleteFirstNode(&head);
  cout << "List after deleting the first node: ";
  display(head);
  deleteLastNode(&head);
  cout << "List after deleting the last node: ";
  display(head);
  deleteAtPosition(&head, 2);
  cout << "List after deleting the node at position 2: ";
```

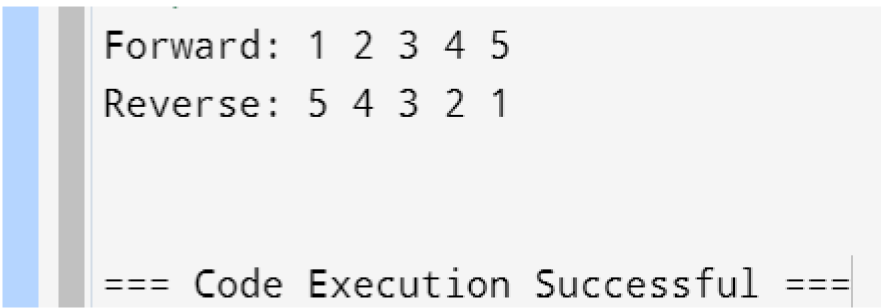


```
Original List: 1 2 3 4 5
List after deleting the first node: 2 3 4 5
List after deleting the last node: 2 3 4
List after deleting the node at position 2: 2 4
```

```
   display(head);
   return 0;}
```

Question 9)

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* prev;
   Node* next;};
void insertAtEnd(Node** head, int data) {
   Node* newNode = new Node();
   newNode->data = data;
   newNode->next = nullptr;
   if (*head == nullptr) {
      newNode->prev = nullptr;
      *head = newNode;
      return;}
   Node* temp = *head;
   while (temp->next != nullptr) {
      temp = temp->next;}
   temp->next = newNode;
   newNode->prev = temp;}
void displayForward(Node* head) {
   if (head == nullptr) {
      cout << "List is empty." << endl;
      return;}
   Node* temp = head;
   cout << "Forward: ";
   while (temp != nullptr) {
      cout << temp->data << " ";
      temp = temp->next;}
   cout << endl;}
void displayReverse(Node* head) {
   if (head == nullptr) {
      cout << "List is empty." << endl;
      return;}
   Node* temp = head;
   while (temp->next != nullptr) {
      temp = temp->next;}
   cout << "Reverse: ";
   while (temp != nullptr) {
      cout << temp->data << " ";
      temp = temp->prev;}
   cout << endl;}
int main() {
   Node* head = nullptr;
   insertAtEnd(&head, 1);
   insertAtEnd(&head, 2);
   insertAtEnd(&head, 3);
   insertAtEnd(&head, 4);
   insertAtEnd(&head, 5);
   displayForward(head);
   displayReverse(head);
   return 0;}
```

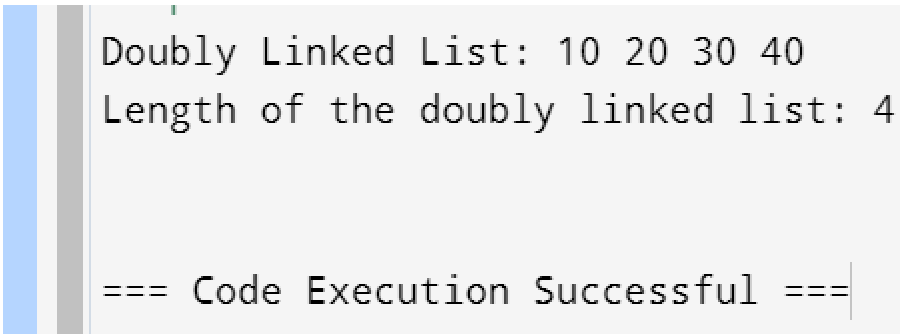


```
Forward: 1 2 3 4 5
Reverse: 5 4 3 2 1


=== Code Execution Successful ===
```

Question 10)

```
#include <iostream>
using namespace std;
struct Node {
  int data;
  Node* prev;
  Node* next;};
void insertAtEnd(Node** head, int data) {
  Node* newNode = new Node();
  newNode->data = data;
  newNode->next = nullptr;
  if (*head == nullptr) {
    newNode->prev = nullptr;
    *head = newNode;
    return;}
  Node* temp = *head;
  while (temp->next != nullptr) {
    temp = temp->next;}
  temp->next = newNode;
  newNode->prev = temp;}
int getLength(Node* head) {
  int length = 0;
  Node* temp = head;
  while (temp != nullptr) {
    length++;
    temp = temp->next;}
  return length;}
void display(Node* head) {
  if (head == nullptr) {
    cout << "List is empty." << endl;
    return;}
  Node* temp = head;
  while (temp != nullptr) {
    cout << temp->data << " ";
    temp = temp->next;}
  cout << endl;}
int main() {
  Node* head = nullptr;
  insertAtEnd(&head, 10);
  insertAtEnd(&head, 20);
  insertAtEnd(&head, 30);
  insertAtEnd(&head, 40);
  cout << "Doubly Linked List: ";
  display(head);
  int length = getLength(head);
  cout << "Length of the doubly linked list: " << length << endl;
  return 0;}
```



```
Doubly Linked List: 10 20 30 40
Length of the doubly linked list: 4


=== Code Execution Successful ===
```