Assignment – 3

11. Create a single linked list and write a function to reverse it.

Source Code: main()

```
#include <iostream>
#include <cstdlib>
using namespace std;
typedef struct node
{
   int data;
   node *next;
} node;
node *head = NULL;
node *createNode(int);
void insertNode(int);
void displayList();
void freeList();
void reverseList();
int main()
{
   int choice, value;
   while (true)
   {
      cout << "\nMenu:\n";
      cout << "1. Insert Node\n";
      cout << "2. Reverse List\n";
      cout << "0. Exit\n";
      displayList();
      cout << "Enter your choice: ";
      cin >> choice;
      switch (choice)
      {
      case 1:
         cout << "Enter value to insert: ";
         cin >> value;
         insertNode(value);
         break;
      case 2:
         reverseList();
         break;
      case 0:
         freeList();
         cout << "Exiting...\n"
              << endl;
         exit(0);
      default:
         cout << "Invalid choice. Try again." << endl;
      }
   }
   return 0;
}
```

Source Code: *createNode(data)

```
node *createNode(int data)
{
   node *newNode = new node();
   newNode->data = data;
   newNode->next = NULL;

   return newNode;
}
```

Source Code: insertNode(data)

```
void insertNode(int data)
{
   node *p, *temp = createNode(data);
   if (head == NULL)
      head = temp;
   else
   {
      p = head;
      while (p->next != NULL)
         p = p->next;
      p->next = temp;
   }
}
```

Source Code: displayList()

```
void displayList()
{
   if (!head)
   {
      cout << "\n\tList is empty!\n"
           << endl;
      return;
   }
   node *p = head;
   cout << "\n\tList is: ";
   while (p != NULL)
   {
      cout << p->data << "->";
      p = p->next;
   }
   cout << "NULL\n"
        << endl;
}
```

Source Code: reverseList()

```
void reverseList()
{
   if (head == NULL || head->next == NULL)
      return;

   node *prevNode = head,
       *currNode = prevNode->next;

   while (currNode != NULL)
   {
      node *nextNode = currNode->next;
      currNode->next = prevNode;

      // Updated
      prevNode = currNode;
      currNode = nextNode;
   }

   head->next = NULL;
   head = prevNode;
}
```

Source Code: freeList()

```
void freeList()
{
   node *temp;
   while (head != nullptr)
   {
      temp = head;
      head = head->next;
      delete temp;
   }
   cout << "\nAll nodes freed." << endl;
}
```

Output

```
Menu:
1. Insert Node
2. Reverse List
0. Exit

    List is empty!

Enter your choice: 1
Enter value to insert: 10

Menu:
1. Insert Node
2. Reverse List
0. Exit

    List is: 10→NULL

Enter your choice: 1
Enter value to insert: 20

Menu:
1. Insert Node
2. Reverse List
0. Exit

    List is: 10→20→NULL

Enter your choice: 1
Enter value to insert: 30

Menu:
1. Insert Node
2. Reverse List
0. Exit

    List is: 10→20→30→NULL

Enter your choice: 2

Menu:
1. Insert Node
2. Reverse List
0. Exit

    List is: 30→20→10→NULL

Enter your choice: 0
All nodes freed.
Exiting...
```

12. Implement a program to detect if a linked list has a cycle.

Source Code: main()

```
#include <iostream>
using namespace std;

typedef struct node
{
   int data;
   node *next;
} node;
node *head = NULL;
node *createNode(int);
node *checkCycle(); // Function to detect and handle cycle (to be implemented
by you)
void createCyclicList();
void displayList();
void freeList();

int main()
{
   createCyclicList(); // Create a hardcoded list with a cycle
   displayList();      // Display the list

   if (!checkCycle()) // Call the function to detect a cycle
      cout << "\n\tIn this list cycle not present!"
           << endl;
   else
      cout << "\n\tIn this list cycle present!"
           << endl;

   freeList(); // Cleanup
   cout << "\tExiting...\n"
        << endl;
   return 0;
}
```

Source Code: *createNode(data)

```
node *createNode(int data)
{
   node *newNode = new node();
   newNode->data = data;
   newNode->next = NULL;

   return newNode;
}
```

Source Code: createCyclicList()

```
void createCyclicList()
{
   // Create nodes
   node *node1 = createNode(1);
   node *node2 = createNode(2);
   node *node3 = createNode(3);
   node *node4 = createNode(4);
   node *node5 = createNode(5);

   // Link nodes to form a list
   head = node1;
   node1->next = node2;
   node2->next = node3;
   node3->next = node4;
   node4->next = node5;

   // Create a cycle (node5 points back to node2)
   node5->next = node2; // Change to node2 to create a cycle
}
```

Source Code: displayList()

```
void displayList()
{
   if (!head)
   {
      cout << "\n\tList is empty!\n"
           << endl;
      return;
   }

   node *p = head;
   int count = 0; // Prevent infinite loop for cyclic list
   cout << "\n\tList is: ";
   while (p != NULL && count < 12)
   {
      cout << p->data << "->";
      p = p->next;
      count++;
   }
   if (count == 12)
      cout << "...\n";
   else
      cout << "NULL\n";
}
```

Source Code: freeList()

```
void freeList()
{
    if (!head)
    {
        cout << "\n\tList is already empty." << endl;
        return;
    }
    // Use checkCycle to detect the meeting point if there's a cycle
    node *meetingPoint = checkCycle();
    if (meetingPoint != NULL) // If a cycle exists
    {
        node *slow = head;
        // Find the start of the cycle
        while (slow->next != meetingPoint->next)
        {
            slow = slow->next;
            meetingPoint = meetingPoint->next;
        }
        // Break the cycle
        meetingPoint->next = NULL;
    }
    // Free the list nodes
    node *temp;
    while (head != nullptr)
    {
        temp = head;
        head = head->next;
        delete temp;
    }

    cout << "\n\tAll nodes freed." << endl;
}
```

Source Code: *checkCycle()

```
// Placeholder function for cycle detection
node *checkCycle()
{
    node *turtle = head, *rabbit = head;
    while (rabbit != NULL && rabbit->next != NULL)
    {
        turtle = turtle->next;
        rabbit = rabbit->next->next;

        if (turtle == rabbit)
            return turtle;
    }
    return NULL;
}
```

Output

```
List is: 1->2->3->4->5->2->3->4->5->2->3->4->...

In this list cycle present!

All nodes freed.
Exiting...
```

Output

```
List is: 1->2->3->4->5->NULL

In this list cycle not present!

All nodes freed.
Exiting...
```

13. Write a function to merge two sorted linked list into a single linked list.

Source Code: main()

```
#include <iostream>
using namespace std;

typedef struct node
{
   int data;
   node *next;
} node;

node *head1 = NULL;
node *head2 = NULL;

node *createNode(int);
node *createList();
node *mergeList();
void displayList(node *);
void freeList(node *);

int main()
{
   cout << "\nFor the 1st list\n";
   cout << "-----------------\n";
   head1 = createList();

   cout << "\nFor the 2nd list\n";
   cout << "-----------------\n";
   head2 = createList();

   cout << '\n';

   cout << "1st List: ";
   displayList(head1);
   cout << "2nd List: ";
   displayList(head2);
   cout << "Merge List: ";
   node *mergedList = mergeList();
   displayList(mergedList);

   freeList(mergedList); // Cleanup
   cout << "Exiting...\n";

   return 0;
}
```

Source Code: *createNode(data)

```
node *createNode(int data)
{
   node *newNode = new node();
   newNode->data = data;
   newNode->next = NULL;

   return newNode;
}
```

Source Code: displayList()

```
void displayList(node *head)
{
   if (!head)
   {
      cout << "Empty!\n"
           << endl;
      return;
   }
   node *p = head;
   while (p != NULL)
   {
      cout << p->data << "->";
      p = p->next;
   }
   cout << "NULL\n"
        << endl;
}
```

Source Code: freeList()

```
void freeList()
{
   node *temp;
   while (head1 != nullptr)
   {
      temp = head1;
      head1 = head1->next;
      delete temp;
   }
   while (head2 != nullptr)
   {
      temp = head2;
      head2 = head2->next;
      delete temp;
   }
   cout << "\nAll nodes freed." << endl;
}
```

Source Code: *mergeList()

```
node *mergeList()
{
   node *p1 = head1, *p2 = head2;
   node *dummy = new node(), *p3 = dummy;

   while (p1 != NULL && p2 != NULL)
   {
      if (p1->data < p2->data)
      {
         p3->next = p1;
         p1 = p1->next;
      }
      else
      {
         p3->next = p2;
         p2 = p2->next;
      }
      p3 = p3->next;
   }

   while (p1 != NULL)
   {
      p3->next = p1;
      p1 = p1->next;
      p3 = p3->next;
   }

   while (p2 != NULL)
   {
      p3->next = p2;
      p2 = p2->next;
      p3 = p3->next;
   }

   return dummy->next;
}
```

Source Code: *createList()

```
node *createList()
{
    int n, value;
    cout << "Enter the number of nodes: ";
    cin >> n;

    if (n <= 0)
    {
        cout << "List size must be greater than 0.\n";
        return NULL;
    }

    // Create the head node
    cout << "Enter value for node 1: ";
    cin >> value;
    node *head = createNode(value);
    node *current = head;

    // Create the remaining nodes
    for (int i = 2; i <= n; ++i)
    {
        cout << "Enter value for node " << i << ": ";
        cin >> value;
        current->next = createNode(value);
        current = current->next;
    }

    return head;
}
```

Output

```
For the 1st list
-----------------
Enter the number of nodes: 5
Enter value for node 1: 1
Enter value for node 2: 2
Enter value for node 3: 4
Enter value for node 4: 6
Enter value for node 5: 8

For the 2nd list
-----------------
Enter the number of nodes: 4
Enter value for node 1: 3
Enter value for node 2: 5
Enter value for node 3: 7
Enter value for node 4: 9
```

Output

1st List: 1->2->4->6->8->NULL

2nd List: 3->5->7->9->NULL

Merge List: 1->2->3->4->5->6->7->8->9->NULL

All nodes freed.
Exiting...

14. Implement an algorithm to find the N^{th} node from the end of a linked list.

Source Code: main()

```
#include <iostream>
using namespace std;

typedef struct Node
{
   int data;
   Node *next;

   Node(int data, Node *next = NULL)
   {
      this->data = data;
      this->next = next;
   }
} Node;

Node *head = NULL;

Node *createNode(int);
Node *findNthFromEnd(int);
void createList();
void displayList();
void freeList();

int main()
{
   cout << '\n';

   createList();
   displayList();

   int n;
   cout << "\tEnter the n'th node: ";
   cin >> n;

   Node *result = findNthFromEnd(n);
   if (result)
      cout << "\tThe " << n << "'th node from end of list is: "
          << result->data << endl;
   else
      cout << "Invalid value of n!" << endl;

   freeList(); // Cleanup
   cout << "\tExiting...\n\n";

   return 0;
}
```

Source Code: *createNode(data)

```
Node *createNode(int data)
{
   return new Node(data, NULL);
}
```

Source Code: displayList()

```
void displayList()
{
   if (!head)
   {
      cout << "\n\tList is empty!\n"
          << endl;
      return;
   }
   Node *p = head;
   cout << "\n\tList is: ";
   while (p != NULL)
   {
      cout << p->data << "->";
      p = p->next;
   }
   cout << "NULL\n"
       << endl;
}
```

Source Code: freeList()

```
void freeList()
{
   Node *temp;
   while (head != nullptr)
   {
      temp = head;
      head = head->next;
      delete temp;
   }
   cout << "\n\tAll nodes freed." << endl;
}
```

Source Code: createList()

```
void createList()
{
   // Create Nodes
   Node *node1 = createNode(1);
   Node *node2 = createNode(2);
   Node *node3 = createNode(3);
   Node *node4 = createNode(4);
   Node *node5 = createNode(5);

   // Link nodes to form a list
   head = node1;
   node1->next = node2;
   node2->next = node3;
   node3->next = node4;
   node4->next = node5;

   node5->next = NULL;
}
```

Source Code: *findFromEnd(n)

```
Node *findNthFromEnd(int n)
{
   Node *slow = head, *fast = head;

   for (int i = 0; i < n; i++)
   {
      if (fast == NULL)
         return NULL;
      fast = fast->next;
   }

   while (fast != NULL)
   {
      slow = slow->next;
      fast = fast->next;
   }

   return slow;
}
```

Output

```
List is: 1->2->3->4->5->NULL

Enter the n'th node: 2
The 2'th node from end of list is: 4

All nodes freed.
Exiting...
```

Output

```
List is: 1->2->3->4->5->NULL

Enter the n'th node: -1

Invalid value of n!

All nodes freed.
Exiting...
```

15. Create a program to delete a node with a given value from a linked list.

Source Code: main()

```
#include <iostream>
#include <cstdlib>
using namespace std;
typedef struct Node
{
   int data;
   Node *next;
} Node;
Node *head = NULL;
Node *createNode(int);
void insertNode(int);
void displayList();
void freeList();
void deleteNodeByValue();
int main()
{
   int choice, value;
   while (true)
   {
      cout << "\nMenu:\n";
      cout << "1. Insert Node\n";
      cout << "2. Delete By Value\n";
      cout << "0. Exit\n";
      displayList();
      cout << "Enter your choice: ";
      cin >> choice;
      switch (choice)
      {
      case 1:
         cout << "Enter value to insert: ";
         cin >> value;
         insertNode(value);
         break;
      case 2:
         deleteNodeByValue();
         break;
      case 0:
         freeList();
         cout << "Exiting...\n"
              << endl;
         exit(0);
      default:
         cout << "Invalid choice. Try again." << endl;
      }
   }
   return 0;
}
```

Source Code: *createNode(data)

```
Node *createNode(int data)
{
   Node *newNode = new Node();
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}
```

Source Code: displayList()

```
void displayList()
{
   if (!head)
   {
      cout << "\n\tList is empty!\n"
           << endl;
      return;
   }
   Node *p = head;
   cout << "\n\tList is: ";
   while (p != NULL)
   {
      cout << p->data << "->";
      p = p->next;
   }
   cout << "NULL\n"
        << endl;
}
```

Source Code: freeList()

```
void freeList()
{
   Node *temp;
   while (head != nullptr)
   {
      temp = head;
      head = head->next;
      delete temp;
   }
   cout << "\nAll nodes freed." << endl;
}
```

Source Code: deleteNodeByValue()

```
void deleteNodeByValue()
{
   int value;
   cout << "Enter the value of the node to delete: ";
   cin >> value;

   if (head == NULL)
   {
      cout << "\n\tThe list is empty. Nothing to delete.\n"
          << endl;
      return;
   }

   // Special case: if the head node contains the value
   if (head->data == value)
   {
      Node *temp = head;
      head = head->next; // Move head to the next node
      free(temp);        // Free the old head
      // cout << "Node with value " << value << " deleted from the head." <<
endl;
      return;
   }

   // General case: traverse the list to find and delete the node
   Node *currNode = head->next; // Start from the second node
   Node *prevNode = head;       // Previous node starts as head

   while (currNode != NULL)
   {
      if (currNode->data == value)
      {
         prevNode->next = currNode->next; // Bypass the current node
         free(currNode);                  // Free the node
         // cout << "Node with value " << value << " deleted." << endl;
         return;
      }
      // Move to the next pair of nodes
      prevNode = currNode;
      currNode = currNode->next;
   }

   // If no node is found with the given value
   cout << "\n\tNode with value " << value << " not found in the list.\n"
       << endl;
}
```

Output

```
Menu:
1. Insert Node
2. Delete By Value
0. Exit

    List is empty!

Enter your choice: 1
Enter value to insert: 10

Menu:
1. Insert Node
2. Delete By Value
0. Exit

    List is: 10->NULL

Enter your choice: 20
Invalid choice. Try again.

Menu:
1. Insert Node
2. Delete By Value
0. Exit

    List is: 10->NULL

Enter your choice: 1
Enter value to insert: 20

Menu:
1. Insert Node
2. Delete By Value
0. Exit

    List is: 10->20->NULL

Enter your choice: 2
Enter the value of the node to delete: 30

    Node with value 30 not found in the list.

Menu:
1. Insert Node
2. Delete By Value
0. Exit

    List is: 10->20->NULL

Enter your choice: 2
Enter the value of the node to delete: 20
```

Output

```
Menu:
1. Insert Node
2. Delete By Value
0. Exit

    List is: 10->NULL

Enter your choice: 2
Enter the value of the node to delete: 10

Menu:
1. Insert Node
2. Delete By Value
0. Exit

    List is empty!

Enter your choice: 2
Enter the value of the node to delete: 10

    The list is empty. Nothing to delete.


Menu:
1. Insert Node
2. Delete By Value
0. Exit

    List is empty!

Enter your choice: 0

All nodes freed.
Exiting...
```

16. Write a function to check if 2 linked list intersect and if they do find intersection node.

Source Code: main()

```
#include <iostream>
using namespace std;

typedef struct Node
{
   int data;
   Node *next;

   Node(int data, Node *next = NULL)
   {
      this->data = data;
      this->next = next;
   }
} Node;

Node *head1 = NULL, *head2 = NULL;

Node *findIntersect();
void createList();
void displayList(Node *);
void freeList();

int main()
{
   createList();
   cout << "\n1st List: ";
   displayList(head1);
   cout << "2nd List: ";
   displayList(head2);

   Node *intersect = findIntersect();

   if (intersect)
      cout << "The intersect node is: " << intersect->data << '\n'
          << endl;
   else
      cout << "\tNo intersection between the two lists." << endl;

   freeList(); // Cleanup
   cout << "Exiting...\n";

   return 0;
}
```

Source Code: displayList()

```
void displayList(Node *head)
{
    if (!head)
    {
        cout << "Empty!\n"
             << endl;
        return;
    }
    Node *p = head;
    while (p != NULL)
    {
        cout << p->data << "->";
        p = p->next;
    }
    cout << "NULL\n"
         << endl;
}
```

Source Code: freeList()

```
void freeList()
{
    Node *temp;
    while (head1 != nullptr)
    {
        temp = head1;
        head1 = head1->next;
        delete temp;
    }
    while (head2 != nullptr)
    {
        temp = head2;
        head2 = head2->next;
        delete temp;
    }
    cout << "\nAll nodes freed." << endl;
}
```

Source Code: createList()

```
void createList()
{
   // 1st list nodes
   Node *node1 = new Node(1);
   Node *node2 = new Node(2);
   Node *node3 = new Node(3);
   Node *node4 = new Node(4);
   Node *node5 = new Node(5);

   // 2nd List nodes
   Node *node6 = new Node(6);
   Node *node7 = new Node(7);
   Node *node8 = new Node(8);

   // Link nodes to form a list
   head1 = node1;
   node1->next = node2;
   node2->next = node3;
   node3->next = node4;
   node4->next = node5;

   head2 = node6;
   node6->next = node7;
   node7->next = node8;

   node8->next = node3;
   node5->next = NULL;
}
```

Source Code: *findIntersect()

```
Node *findIntersect()
{
   Node *p1 = head1, *p2 = head2;

   // Traverse both lists. When one pointer reaches the end, switch to the other
list.
   while (p1 != p2)
   {
      p1 = (p1 == NULL) ? head2 : p1->next;
      p2 = (p2 == NULL) ? head1 : p2->next;
   }

   // Either intersection node or NULL (if no intersection)
   return p1;
}
```

Output

1st List: 1->2->3->4->5->NULL

2nd List: 6->7->8->3->4->5->NULL

The intersect node is: 3

17. Implement a function to add 2 numbers represented by linked lists (e.g., 342 + 465 = 807).

Source Code: main()

```
#include <iostream>
using namespace std;
// Node structure
typedef struct Node
{
   int data;
   Node *next;
   Node(int data, Node *next = NULL)
   {
      this->data = data;
      this->next = next;
   }
} Node;
// Function prototypes
void createList(Node *&head);
void displayList(Node *head);
void freeList(Node *head);
void insertAtTail(Node *&head, Node *&tail, int data);
Node *reverseList(Node *head);
Node *add(Node *first, Node *second);
Node *addTwoLists(Node *first, Node *second);
int main()
{
   Node *first = NULL, *second = NULL;
   // Create lists
   cout << '\n'
        << "Creating the first list...." << endl;
   createList(first);
   cout << "Creating the second list...." << endl;
   createList(second);
   // Display the lists
   cout << "\nFirst List: ";
   displayList(first);
   cout << "\nSecond List: ";
   displayList(second);
   cout << '\n';
   // Add the two lists
   Node *sum = addTwoLists(first, second);
   // Display the result
   cout << "Sum List: ";
   displayList(sum);
   // Free memory
   freeList(first);
   freeList(second);
   freeList(sum);
   cout << "\nExiting...\n";
   return 0;
}
```

Source Code: displayList()

```
void displayList(Node *head)
{
    if (!head)
    {
        cout << "Empty!\n";
        return;
    }

    Node *p = head;
    while (p != NULL)
    {
        cout << p->data;
        if (p->next != NULL)
            cout << "->";
        p = p->next;
    }
    cout << "->NULL\n";
}
```

Source Code: freeList()

```
void freeList(Node *head)
{
    Node *temp;
    while (head != nullptr)
    {
        temp = head;
        head = head->next;
        delete temp;
    }
    // cout << "All nodes freed." << endl;
}
```

Source Code: createList()

```
void createList(Node *&head, int number)
{
    if (number == 0)
    {
        head = new Node(0); // Handle the case for 0
        return;
    }

    while (number > 0)
    {
        int digit = number % 10; // Extract the last digit
        Node *newNode = new Node(digit);

        newNode->next = head; // Insert at the beginning (most significant digit)
        head = newNode;       // Move the head to the new node

        number /= 10; // Remove the last digit
    }
}
```

Source Code: *reverseList()

```
Node *reverseList(Node *head)
{
    if (head == NULL || head->next == NULL)
        return head;

    Node *prevNode = NULL, *currNode = head;

    while (currNode != NULL)
    {
        Node *nextNode = currNode->next; // Store the next node
        currNode->next = prevNode;       // Reverse the link
        prevNode = currNode;             // Move prevNode forward
        currNode = nextNode;             // Move currNode forward
    }

    return prevNode; // New head of the reversed list
}
```

Source Code: insertAtTail()

```
void insertAtTail(Node *&head, Node *&tail, int data)
{
    Node *temp = new Node(data);
    if (head == NULL)
    {
        head = temp;
        tail = temp;
        return;
    }
    else
    {
        tail->next = temp;
        tail = temp;
    }
}
```

Source Code: *addTwoLists()

```
Node *addTwoLists(Node *first, Node *second)
{
    first = reverseList(first);
    second = reverseList(second);

    Node *result = add(first, second);

    // Reverse the result to maintain proper order
    return reverseList(result);
}
```

Source Code: *add()

```
Node *add(Node *first, Node *second)
{
   int carry = 0;
   Node *ansHead = NULL, *ansTail = NULL;

   while (first != NULL && second != NULL)
   {
      int sum = carry + first->data + second->data;
      int digit = sum % 10;
      insertAtTail(ansHead, ansTail, digit);
      carry = sum / 10;
      first = first->next;
      second = second->next;
   }
   while (first != NULL)    {
      int sum = carry + first->data;
      int digit = sum % 10;
      insertAtTail(ansHead, ansTail, digit);
      carry = sum / 10;
      first = first->next;
   }
   while (second != NULL)    {
      int sum = carry + second->data;
      int digit = sum % 10;
      insertAtTail(ansHead, ansTail, digit);
      carry = sum / 10;
      second = second->next;
   }
   while (carry != 0)    {
      int sum = carry;
      int digit = sum % 10;
      insertAtTail(ansHead, ansTail, digit);
      carry = sum / 10;
   }

   return ansHead;
}
```

Output

```
First List: 3->4->5->NULL

Second List: 4->5->NULL

Sum List: 3->9->0->NULL

Exiting...
```