National University of Computer and Emerging Sciences



Name: Muhammad Suleman

Roll #: 22F-3350

Section: BCS-4B

Lab # 03

# Problem 01

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int data)

: data(data), next(nullptr) {}

};

class LinkedList

{

public:

Node \*head;

LinkedList()

: head(nullptr) {}

void insertAtBeginning(int data)

{

Node \*current = new Node(data);

current->next = head;

head = current;

}

void insertAtEnd(int data)

{

Node \*newNode = new Node(data);

// if list was empty

if (head == nullptr)

{

head = newNode;

return;

}

Node \*current = head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = newNode;

}

void insertAtPosition(int data, int position)

{

if (position <= 0)

{

cout << "Invalid position: " << position << endl;

return;

}

if (position == 1)

{

insertAtBeginning(data);

return;

}

Node \*newNode = new Node(data);

Node \*current = head;

int count = 1;

while (current != nullptr && count < position - 1)

{

current = current->next;

count++;

}

if (current == nullptr)

{

cout << "Invalid position: " << position << endl;

return;

}

newNode->next = current->next;

current->next = newNode;

}

void printList()

{

if (head == nullptr)

{

cout << "List is empty" << endl;

return;

}

Node \*current = head;

while (current != nullptr)

{

cout << current->data << " -> ";

current = current->next;

}

cout << "nullptr" << endl;

}

};

int main()

{

LinkedList list;

list.insertAtBeginning(10);

list.insertAtBeginning(15);

list.insertAtBeginning(20);

list.insertAtEnd(25);

list.insertAtEnd(30);

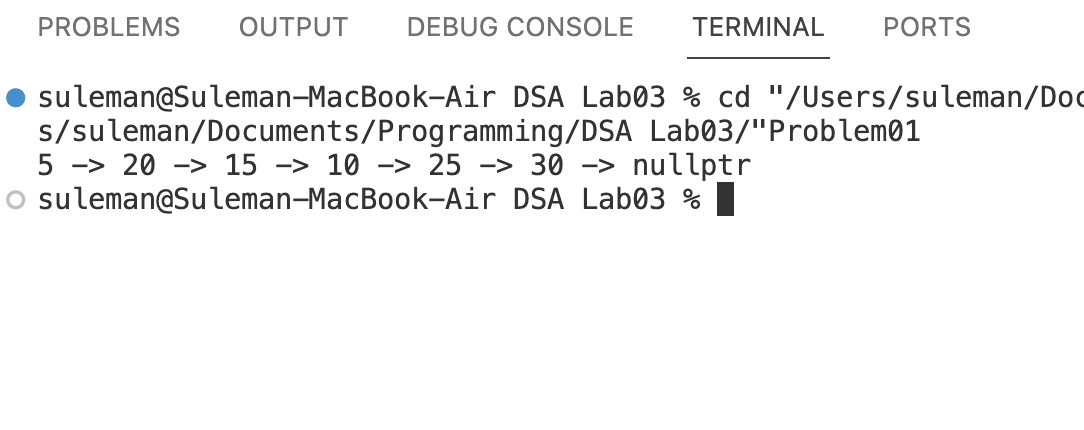
list.insertAtPosition(5, 1);

list.printList();

return 0;

}

## Output



# Problem 02

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int data) : data(data), next(nullptr) {}

};

class LinkedList

{

public:

Node \*head;

LinkedList() : head(nullptr) {}

void insertAtBeginning(int data)

{

Node \*current = new Node(data);

current->next = head;

head = current;

}

void insertAtEnd(int data)

{

Node \*newNode = new Node(data);

if (head == nullptr)

{

head = newNode;

return;

}

Node \*current = head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = newNode;

}

void insertAtPosition(int data, int position)

{

if (position <= 0)

{

cout << "Invalid position: " << position << endl;

return;

}

if (position == 1)

{

insertAtBeginning(data);

return;

}

Node \*newNode = new Node(data);

Node \*current = head;

int count = 1;

while (current != nullptr && count < position - 1)

{

current = current->next;

count++;

}

if (current == nullptr)

{

cout << "Invalid position: " << position << endl;

return;

}

newNode->next = current->next;

current->next = newNode;

}

void printList()

{

if (head == nullptr)

{

cout << "List is empty" << endl;

return;

}

Node \*current = head;

while (current != nullptr)

{

cout << current->data << " -> ";

current = current->next;

}

cout << "nullptr" << endl;

}

bool isPalindrome()

{

if (head == nullptr || head->next == nullptr)

{

return true;

}

Node \*slow = head;

Node \*fast = head;

while (fast->next != nullptr && fast->next->next != nullptr)

{

slow = slow->next;

fast = fast->next->next;

}

Node \*secondHalf = reverseList(slow->next);

Node \*firstHalf = head;

while (secondHalf != nullptr)

{

if (firstHalf->data != secondHalf->data)

{

return false;

}

firstHalf = firstHalf->next;

secondHalf = secondHalf->next;

}

return true;

}

private:

Node \*reverseList(Node \*start)

{

Node \*prev = nullptr;

Node \*current = start;

Node \*next = nullptr;

while (current != nullptr)

{

next = current->next;

current->next = prev;

prev = current;

current = next;

}

return prev;

}

};

int main()

{

LinkedList list;

list.insertAtBeginning(10);

list.insertAtBeginning(15);

list.insertAtBeginning(20);

list.insertAtEnd(25);

list.insertAtEnd(30);

list.insertAtPosition(5, 1);

list.printList();

if (list.isPalindrome())

{

cout << "The linked list is a palindrome" << endl;

}

else

{

cout << "The linked list is not a palindrome" << endl;

}

LinkedList list2;

if (list2.isPalindrome())

{

cout << "The linked list 2 is a palindrome" << endl;

}

else

{

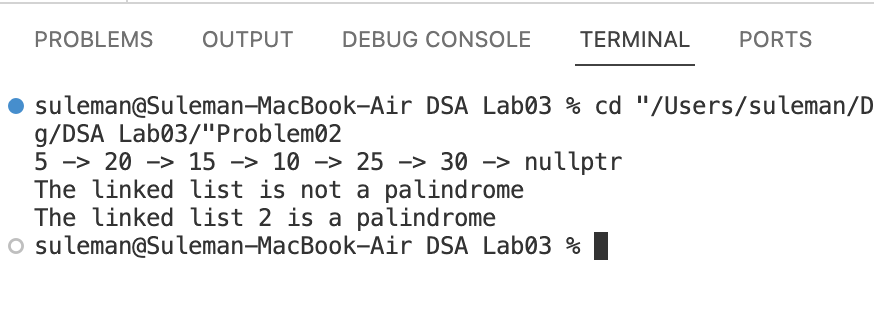
cout << "The linked list 2 is not a palindrome" << endl;

}

return 0;

}

## Output



# Problem 03

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int data)

: data(data), next(nullptr) {}

};

class LinkedList

{

public:

Node \*head;

LinkedList()

: head(nullptr) {}

void insertAtBeginning(int data)

{

Node \*current = new Node(data);

current->next = head;

head = current;

}

void insertAtEnd(int data)

{

Node \*newNode = new Node(data);

// if list was empty

if (head == nullptr)

{

head = newNode;

return;

}

Node \*current = head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = newNode;

}

void insertAtPosition(int data, int position)

{

if (position <= 0)

{

cout << "Invalid position: " << position << endl;

return;

}

if (position == 1)

{

insertAtBeginning(data);

return;

}

Node \*newNode = new Node(data);

Node \*current = head;

int count = 1;

while (current != nullptr && count < position - 1)

{

current = current->next;

count++;

}

if (current == nullptr)

{

cout << "Invalid position: " << position << endl;

return;

}

newNode->next = current->next;

current->next = newNode;

}

void printList()

{

if (head == nullptr)

{

cout << "List is empty" << endl;

return;

}

Node \*current = head;

while (current != nullptr)

{

cout << current->data << " -> ";

current = current->next;

}

cout << "nullptr" << endl;

}

void merge(LinkedList &list1, LinkedList &list2, LinkedList &result)

{

Node \*current1 = list1.head;

Node \*current2 = list2.head;

while (current1 != nullptr && current2 != nullptr)

{

if (current1->data < current2->data)

{

result.insertAtEnd(current1->data);

current1 = current1->next;

}

else

{

result.insertAtEnd(current2->data);

current2 = current2->next;

}

}

while (current1 != nullptr)

{

result.insertAtEnd(current1->data);

current1 = current1->next;

}

while (current2 != nullptr)

{

result.insertAtEnd(current2->data);

current2 = current2->next;

}

}

};

int main()

{

LinkedList list1;

list1.insertAtBeginning(40);

list1.insertAtBeginning(30);

list1.insertAtBeginning(20);

list1.insertAtBeginning(10);

cout << "List 1: ";

list1.printList();

LinkedList list2;

list2.insertAtBeginning(90);

list2.insertAtBeginning(80);

list2.insertAtBeginning(70);

list2.insertAtBeginning(60);

cout << "List 2: ";

list2.printList();

LinkedList resultant;

resultant.merge(list1, list2, resultant);

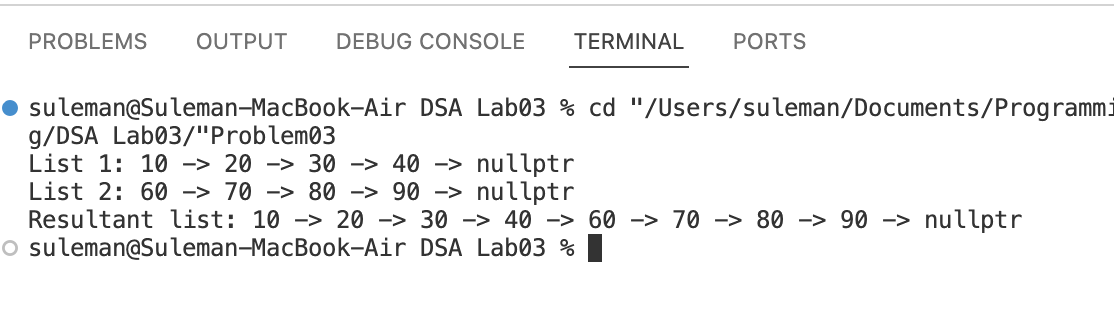
cout << "Resultant list: ";

resultant.printList();

return 0;

}

## Output



# Problem 04

#include <iostream>

using namespace std;

class Node

{

public:

char data;

Node \*next;

Node(char data) : data(data), next(nullptr) {}

};

class LinkedList

{

public:

Node \*head;

LinkedList() : head(nullptr) {}

void insertAtBeginning(char data)

{

Node \*current = new Node(data);

current->next = head;

head = current;

}

void insertAtEnd(char data)

{

Node \*newNode = new Node(data);

if (head == nullptr)

{

head = newNode;

return;

}

Node \*current = head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = newNode;

}

void insertAtPosition(char data, int position)

{

if (position <= 0)

{

cout << "Invalid position: " << position << endl;

return;

}

if (position == 1)

{

insertAtBeginning(data);

return;

}

Node \*newNode = new Node(data);

Node \*current = head;

int count = 1;

while (current != nullptr && count < position - 1)

{

current = current->next;

count++;

}

if (current == nullptr)

{

cout << "Invalid position: " << position << endl;

return;

}

newNode->next = current->next;

current->next = newNode;

}

void printList()

{

if (head == nullptr)

{

cout << "List is empty" << endl;

return;

}

Node \*current = head;

while (current != nullptr)

{

cout << current->data << " -> ";

current = current->next;

}

cout << "nullptr" << endl;

}

LinkedList reverseCopy()

{

LinkedList reversedList;

Node \*current = head;

while (current != nullptr)

{

reversedList.insertAtBeginning(current->data);

current = current->next;

}

return reversedList;

}

};

int main()

{

LinkedList list;

list.insertAtBeginning('A');

list.insertAtBeginning('B');

list.insertAtBeginning('C');

list.insertAtBeginning('D');

list.insertAtBeginning('E');

list.insertAtBeginning('F');

list.insertAtBeginning('G');

list.insertAtBeginning('H');

list.insertAtBeginning('I');

list.insertAtBeginning('J');

LinkedList reversedList = list.reverseCopy();

cout << "Original list: ";

list.printList();

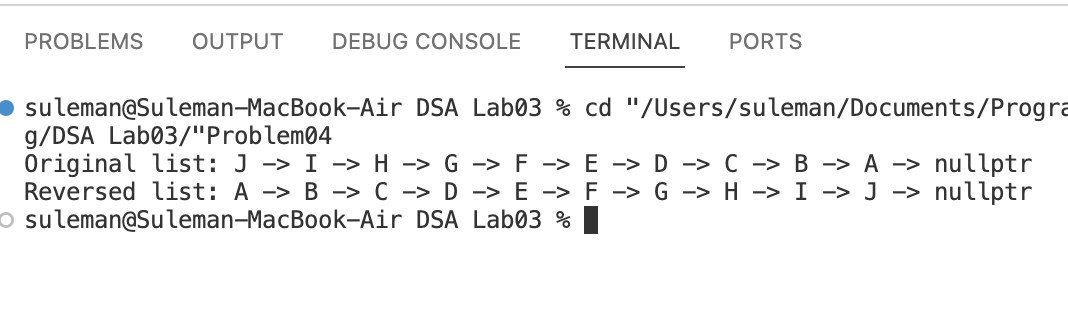
cout << "Reversed list: ";

reversedList.printList();

return 0;

}

## Output



# Problem 05

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int data)

: data(data), next(nullptr) {}

};

class LinkedList

{

public:

Node \*head;

LinkedList()

: head(nullptr) {}

void insertAtBeginning(int data)

{

Node \*current = new Node(data);

current->next = head;

head = current;

}

void insertAtEnd(int data)

{

Node \*newNode = new Node(data);

// if list was empty

if (head == nullptr)

{

head = newNode;

return;

}

Node \*current = head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = newNode;

}

void insertAtPosition(int data, int position)

{

if (position <= 0)

{

cout << "Invalid position: " << position << endl;

return;

}

if (position == 1)

{

insertAtBeginning(data);

return;

}

Node \*newNode = new Node(data);

Node \*current = head;

int count = 1;

while (current != nullptr && count < position - 1)

{

current = current->next;

count++;

}

if (current == nullptr)

{

cout << "Invalid position: " << position << endl;

return;

}

newNode->next = current->next;

current->next = newNode;

}

void printList()

{

if (head == nullptr)

{

cout << "List is empty" << endl;

return;

}

Node \*current = head;

while (current != nullptr)

{

cout << current->data << " -> ";

current = current->next;

}

cout << "nullptr" << endl;

}

void swapFirstAndLast()

{

if (head == nullptr || head->next == nullptr)

{

cout << "List has less than two nodes" << endl;

return;

}

Node \*current = head;

while (current->next != nullptr)

{

current = current->next;

}

swap(head->data, current->data);

}

};

int main()

{

LinkedList list;

list.insertAtBeginning(10);

list.insertAtBeginning(15);

list.insertAtBeginning(20);

list.insertAtEnd(25);

list.insertAtEnd(30);

list.insertAtPosition(5, 1);

list.printList();

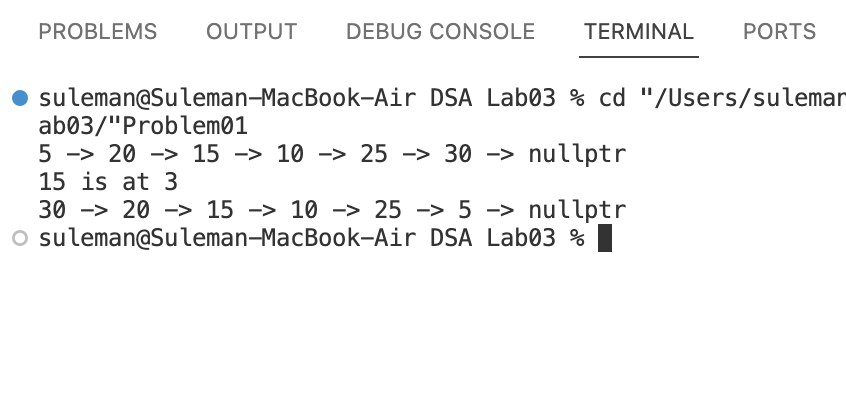
list.swapFirstAndLast();

list.printList();

return 0;

}

## Output



# Problem 06

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int data)

: data(data), next(nullptr) {}

};

class LinkedList

{

public:

Node \*head;

LinkedList()

: head(nullptr) {}

void insertAtBeginning(int data)

{

Node \*current = new Node(data);

current->next = head;

head = current;

}

void insertAtEnd(int data)

{

Node \*newNode = new Node(data);

// if list was empty

if (head == nullptr)

{

head = newNode;

return;

}

Node \*current = head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = newNode;

}

void insertAtPosition(int data, int position)

{

if (position <= 0)

{

cout << "Invalid position: " << position << endl;

return;

}

if (position == 1)

{

insertAtBeginning(data);

return;

}

Node \*newNode = new Node(data);

Node \*current = head;

int count = 1;

while (current != nullptr && count < position - 1)

{

current = current->next;

count++;

}

if (current == nullptr)

{

cout << "Invalid position: " << position << endl;

return;

}

newNode->next = current->next;

current->next = newNode;

}

void printList()

{

if (head == nullptr)

{

cout << "List is empty" << endl;

return;

}

Node \*current = head;

while (current != nullptr)

{

cout << current->data << " -> ";

current = current->next;

}

cout << "nullptr" << endl;

}

void search(int data)

{

if (head == nullptr)

{

cout << "List is empty" << endl;

return;

}

Node \*current = head;

int position = 1;

while (current->data != data && current->next != nullptr)

{

current = current->next;

position++;

}

if (current != nullptr)

{

cout << current->data << " is at " << position << endl;

}

else

{

cout << data << " not present in list" << endl;

}

}

void swapFirstAndLast()

{

if (head == nullptr || head->next == nullptr)

{

cout << "List has less than two nodes" << endl;

return;

}

Node \*current = head;

while (current->next != nullptr)

{

current = current->next;

}

swap(head->data, current->data);

}

};

int main()

{

LinkedList list;

list.insertAtBeginning(10);

list.insertAtBeginning(15);

list.insertAtBeginning(20);

list.insertAtEnd(25);

list.insertAtEnd(30);

list.insertAtPosition(5, 1);

list.printList();

list.search(15);

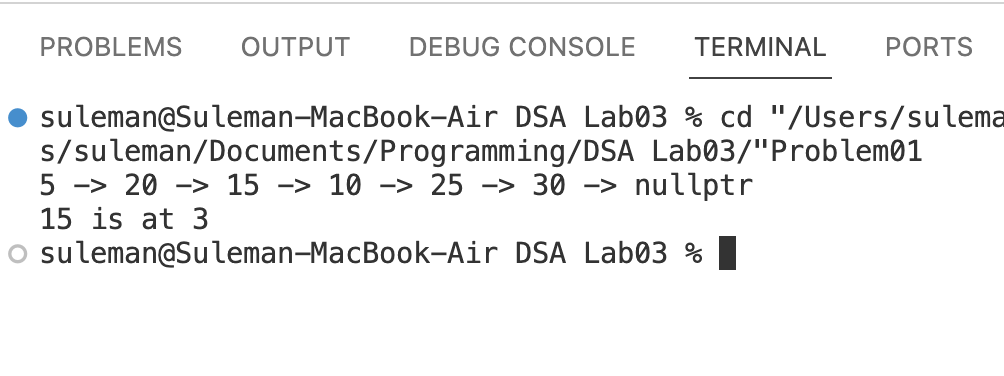
list.swapFirstAndLast();

list.printList();

return 0;

}

## Output



# Problem 07

#include <iostream>

using namespace std;

class Node

{

public:

int data;

Node \*next;

Node(int data)

: data(data), next(nullptr) {}

};

class LinkedList

{

public:

Node \*head;

LinkedList()

: head(nullptr) {}

void insertAtBeginning(int data)

{

Node \*current = new Node(data);

current->next = head;

head = current;

}

void insertAtEnd(int data)

{

Node \*newNode = new Node(data);

// if list was empty

if (head == nullptr)

{

head = newNode;

return;

}

Node \*current = head;

while (current->next != nullptr)

{

current = current->next;

}

current->next = newNode;

}

void insertAtPosition(int data, int position)

{

if (position <= 0)

{

cout << "Invalid position: " << position << endl;

return;

}

if (position == 1)

{

insertAtBeginning(data);

return;

}

Node \*newNode = new Node(data);

Node \*current = head;

int count = 1;

while (current != nullptr && count < position - 1)

{

current = current->next;

count++;

}

if (current == nullptr)

{

cout << "Invalid position: " << position << endl;

return;

}

newNode->next = current->next;

current->next = newNode;

}

void printList()

{

if (head == nullptr)

{

cout << "List is empty" << endl;

return;

}

Node \*current = head;

while (current != nullptr)

{

cout << current->data << " -> ";

current = current->next;

}

cout << "nullptr" << endl;

}

void search(int data)

{

if (head == nullptr)

{

cout << "List is empty" << endl;

return;

}

Node \*current = head;

int position = 1;

while (current->data != data && current->next != nullptr)

{

current = current->next;

position++;

}

if (current != nullptr)

{

cout << current->data << " is at " << position << endl;

}

else

{

cout << data << " not present in list" << endl;

}

}

void swapFirstAndLast()

{

if (head == nullptr || head->next == nullptr)

{

cout << "List has less than two nodes" << endl;

return;

}

Node \*current = head;

while (current->next != nullptr)

{

current = current->next;

}

swap(head->data, current->data);

}

void bubbleSort()

{

if (head == nullptr)

{

cout << "List is empty" << endl;

return;

}

bool swapped;

Node \*ptr1;

Node \*lptr = nullptr;

do

{

swapped = false;

ptr1 = head;

while (ptr1->next != lptr)

{

if (ptr1->data > ptr1->next->data)

{

int temp = ptr1->data;

ptr1->data = ptr1->next->data;

ptr1->next->data = temp;

swapped = true;

}

ptr1 = ptr1->next;

}

lptr = ptr1;

} while (swapped);

}

};

int main()

{

LinkedList list;

list.insertAtBeginning(10);

list.insertAtBeginning(15);

list.insertAtBeginning(20);

list.insertAtEnd(25);

list.insertAtEnd(30);

list.insertAtPosition(5, 1);

list.printList();

list.search(15);

list.swapFirstAndLast();

list.printList();

cout << "Sorted list" << endl;

list.bubbleSort();

list.printList();

return 0;

}

## Output

