Name memansi  
roll no 1024030006  
subgroup 2c35

Ques 1 Develop a menu driven program for the following operations of on a Singly Linked List. (a) Insertion at the beginning. (b) Insertion at the end. (c) Insertion in between (before or after a node having a specific value, say 'Insert a new Node 35 before/after the Node 30'). (d) Deletion from the beginning. (e) Deletion from the end. (f) Deletion of a specific node, say 'Delete Node 60'). (g) Search for a node and display its position from head. (h) Display all the node values.

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL;

void insertAtBeginning(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

newNode->data = data;

newNode->next = head;

head = newNode;

}

void insertAtEnd(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

struct Node\* temp = head;

newNode->data = data;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

} else {

while (temp->next != NULL)

temp = temp->next;

temp->next = newNode;

}

}

void insertBefore(int key, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

struct Node\* temp = head;

struct Node\* prev = NULL;

if (head == NULL) return;

if (head->data == key) {

insertAtBeginning(data);

return;

}

while (temp != NULL && temp->data != key) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) return;

newNode->data = data;

newNode->next = temp;

prev->next = newNode;

}

void insertAfter(int key, int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

struct Node\* temp = head;

while (temp != NULL && temp->data != key)

temp = temp->next;

if (temp == NULL) return;

newNode->data = data;

newNode->next = temp->next;

temp->next = newNode;

}

void deleteFromBeginning() {

if (head == NULL) return;

struct Node\* temp = head;

head = head->next;

free(temp);

}

void deleteFromEnd() {

if (head == NULL) return;

struct Node\* temp = head;

struct Node\* prev = NULL;

if (head->next == NULL) {

free(head);

head = NULL;

return;

}

while (temp->next != NULL) {

prev = temp;

temp = temp->next;

}

prev->next = NULL;

free(temp);

}

void deleteSpecific(int key) {

if (head == NULL) return;

struct Node\* temp = head;

struct Node\* prev = NULL;

if (head->data == key) {

head = head->next;

free(temp);

return;

}

while (temp != NULL && temp->data != key) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) return;

prev->next = temp->next;

free(temp);

}

void searchNode(int key) {

struct Node\* temp = head;

int pos = 1;

while (temp != NULL) {

if (temp->data == key) {

printf("Found at position %d\n", pos);

return;

}

temp = temp->next;

pos++;

}

printf("Not found\n");

}

void displayList() {

struct Node\* temp = head;

if (head == NULL) {

printf("List is empty\n");

return;

}

while (temp != NULL) {

printf("%d -> ", temp->data);

temp = temp->next;

}

printf("NULL\n");

}

int main() {

int choice, data, key;

while (1) {

printf("\n1.Insert at Beginning\n2.Insert at End\n3.Insert Before\n4.Insert After\n5.Delete from Beginning\n6.Delete from End\n7.Delete Specific\n8.Search\n9.Display\n10.Exit\nEnter choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

scanf("%d", &data);

insertAtBeginning(data);

break;

case 2:

scanf("%d", &data);

insertAtEnd(data);

break;

case 3:

scanf("%d", &key);

scanf("%d", &data);

insertBefore(key, data);

break;

case 4:

scanf("%d", &key);

scanf("%d", &data);

insertAfter(key, data);

break;

case 5:

deleteFromBeginning();

break;

case 6:

deleteFromEnd();

break;

case 7:

scanf("%d", &key);

deleteSpecific(key);

break;

case 8:

scanf("%d", &key);

searchNode(key);

break;

case 9:

displayList();

break;

case 10:

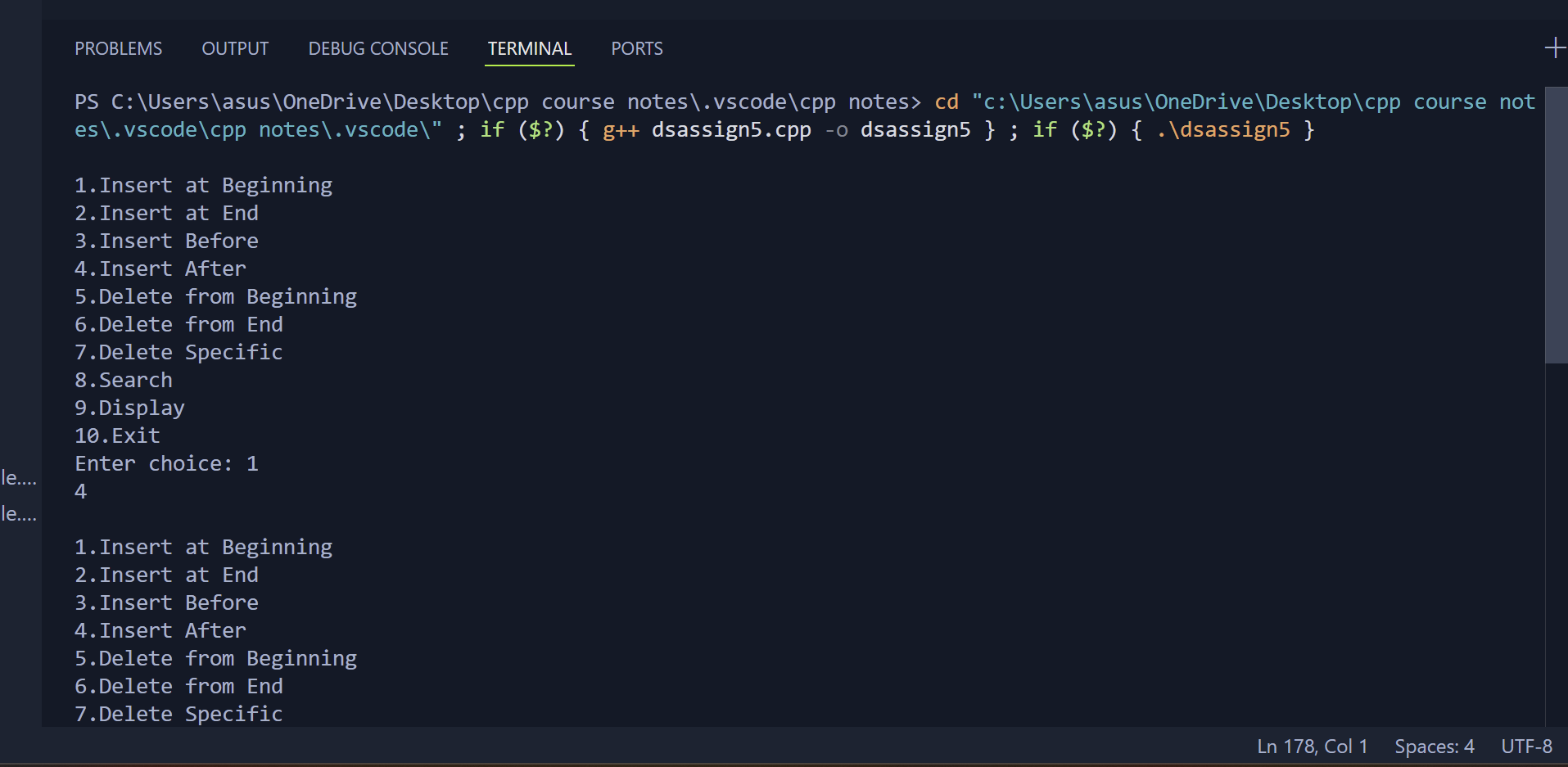
exit(0);

}

}

return 0;

}



2. Write a program to count the number of occurrences of a given key in a singly linked list and then delete all the occurrences. For example, if given linked list is 1->2->1- >2->1->3->1 and given key is 1, then output should be 4. After deletion of all the occurrences of 1, the linked list is 2->2->3.

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

Node\* head = nullptr;

void insertAtEnd(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr)

temp = temp->next;

temp->next = newNode;

}

}

int countOccurrences(int key) {

int count = 0;

Node\* temp = head;

while (temp != nullptr) {

if (temp->data == key)

count++;

temp = temp->next;

}

return count;

}

void deleteAllOccurrences(int key) {

Node\* temp = head;

Node\* prev = nullptr;

while (temp != nullptr && temp->data == key) {

head = temp->next;

delete temp;

temp = head;

}

while (temp != nullptr) {

while (temp != nullptr && temp->data != key) {

prev = temp;

temp = temp->next;

}

if (temp == nullptr) return;

prev->next = temp->next;

delete temp;

temp = prev->next;

}

}

void displayList() {

Node\* temp = head;

if (head == nullptr) {

cout << "List is empty\n";

return;

}

while (temp != nullptr) {

cout << temp->data << " -> ";

temp = temp->next;

}

cout << "NULL\n";

}

int main() {

int n, data, key, count;

cout << "Enter number of elements: ";

cin >> n;

cout << "Enter " << n << " elements:\n";

for (int i = 0; i < n; i++) {

cin >> data;

insertAtEnd(data);

}

cout << "Linked List before deletion:\n";

displayList();

cout << "Enter key to count and delete: ";

cin >> key;

count = countOccurrences(key);

cout << "Occurrences of " << key << " = " << count << endl;

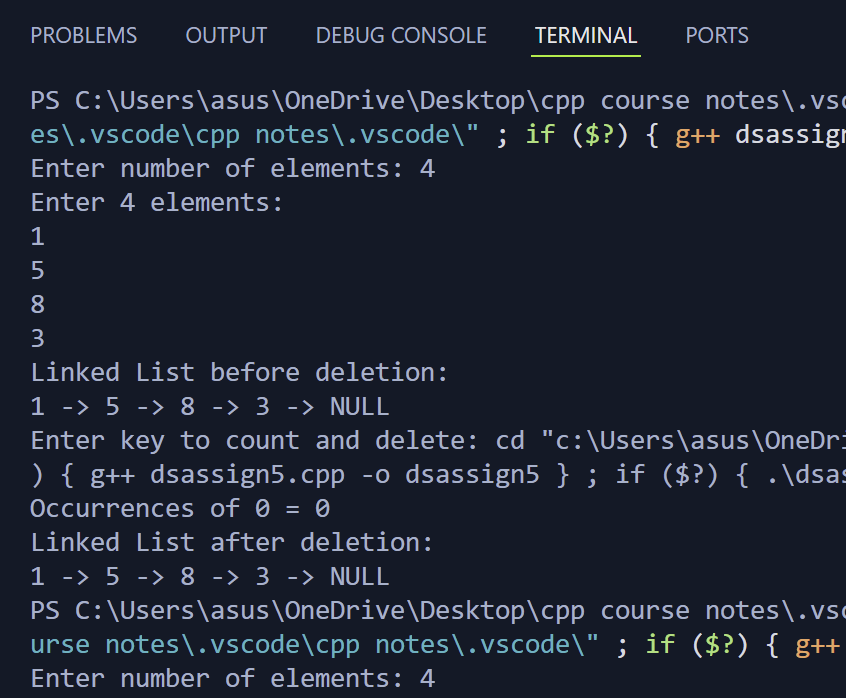
deleteAllOccurrences(key);

cout << "Linked List after deletion:\n";

displayList();

return 0;

}



3. Write a program to find the middle of a linked list. Input: 1->2->3->4->5 Output- 3

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

Node\* head = nullptr;

void insertAtEnd(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr)

temp = temp->next;

temp->next = newNode;

}

}

int findMiddle() {

if (head == nullptr) {

cout << "List is empty\n";

return -1;

}

Node\* slow = head;

Node\* fast = head;

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

slow = slow->next; // move by one

fast = fast->next->next; // move by two

}

return slow->data; // slow will be at middle

}

void displayList() {

Node\* temp = head;

if (head == nullptr) {

cout << "List is empty\n";

return;

}

while (temp != nullptr) {

cout << temp->data << " -> ";

temp = temp->next;

}

cout << "NULL\n";

}

int main() {

int n, data;

cout << "Enter number of elements: ";

cin >> n;

cout << "Enter " << n << " elements:\n";

for (int i = 0; i < n; i++) {

cin >> data;

insertAtEnd(data);

}

cout << "Linked List:\n";

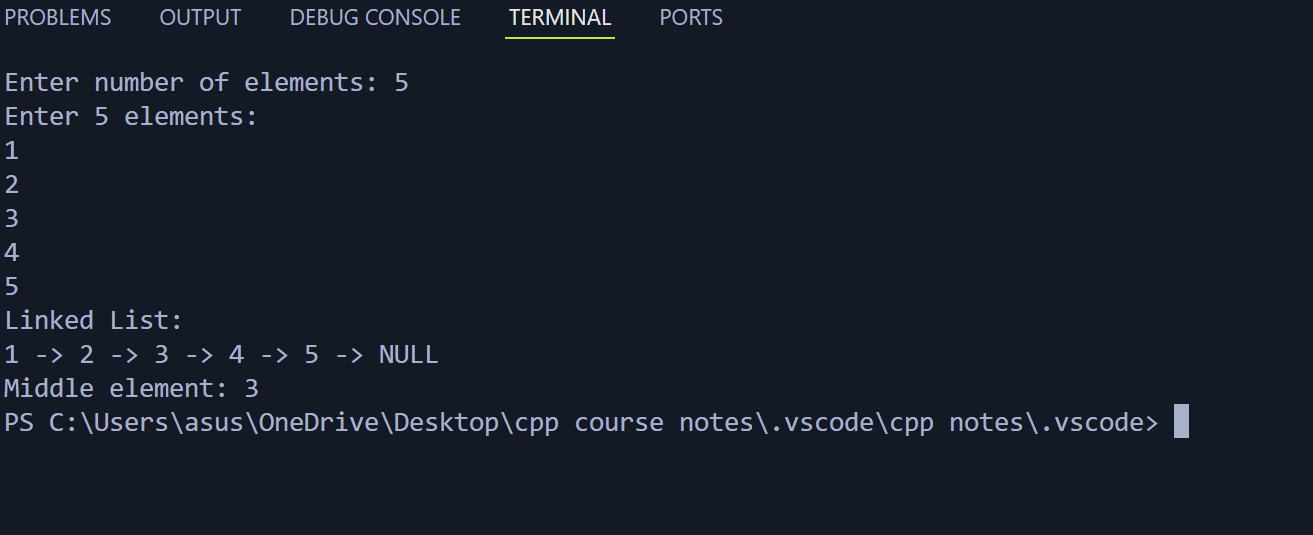
displayList();

int middle = findMiddle();

if (middle != -1)

cout << "Middle element: " << middle << endl;

return 0;

}  


4. Write a program to reverse a linked list. Input: 1->2->3->4->NULL Output: 4->3->2->1->NULL  
#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

Node\* head = nullptr;

void insertAtEnd(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr)

temp = temp->next;

temp->next = newNode;

}

}

void reverseList() {

Node\* prev = nullptr;

Node\* current = head;

Node\* next = nullptr;

while (current != nullptr) {

next = current->next; // store next node

current->next = prev; // reverse pointer

prev = current; // move prev forward

current = next; // move current forward

}

head = prev; // new head after reversing

}

void displayList() {

Node\* temp = head;

if (head == nullptr) {

cout << "List is empty\n";

return;

}

while (temp != nullptr) {

cout << temp->data << " -> ";

temp = temp->next;

}

cout << "NULL\n";

}

int main() {

int n, data;

cout << "Enter number of elements: ";

cin >> n;

cout << "Enter " << n << " elements:\n";

for (int i = 0; i < n; i++) {

cin >> data;

insertAtEnd(data);

}

cout << "Linked List before reversing:\n";

displayList();

reverseList();

cout << "Linked List after reversing:\n";

displayList();

return 0;

}

