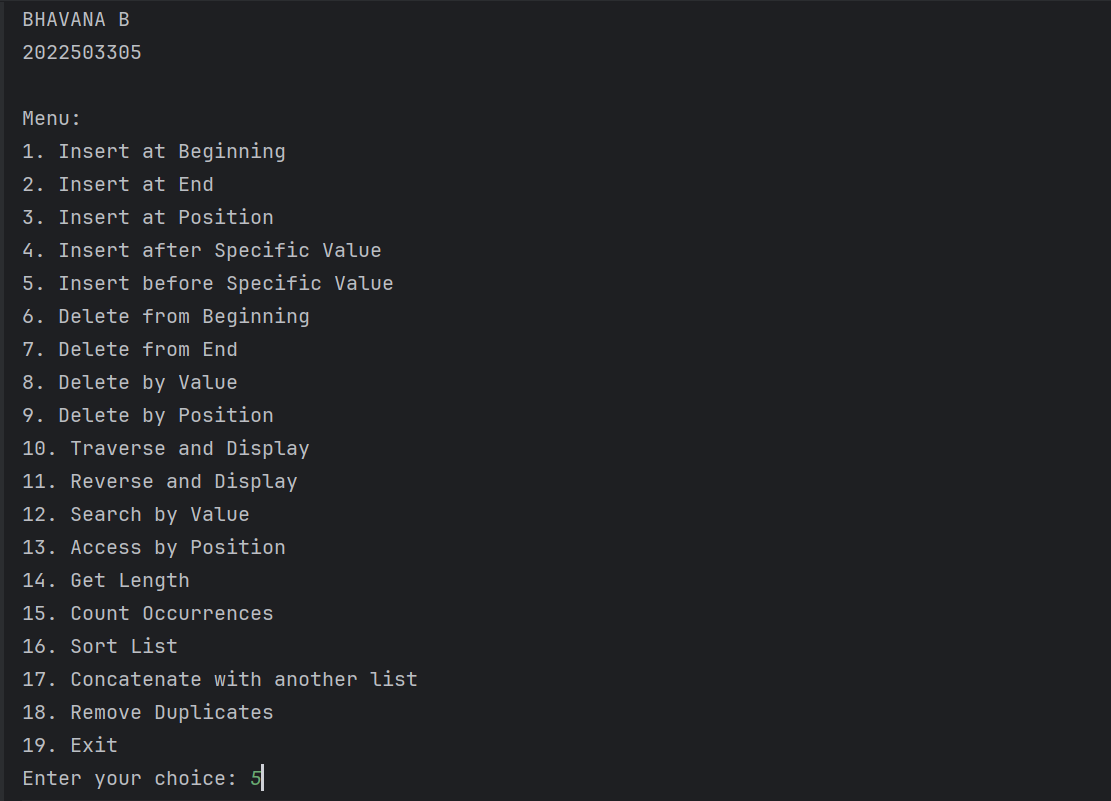
**EXERCISE 1: Write a Java Program to Implement The Below Listed Task**

**PROGRAM**

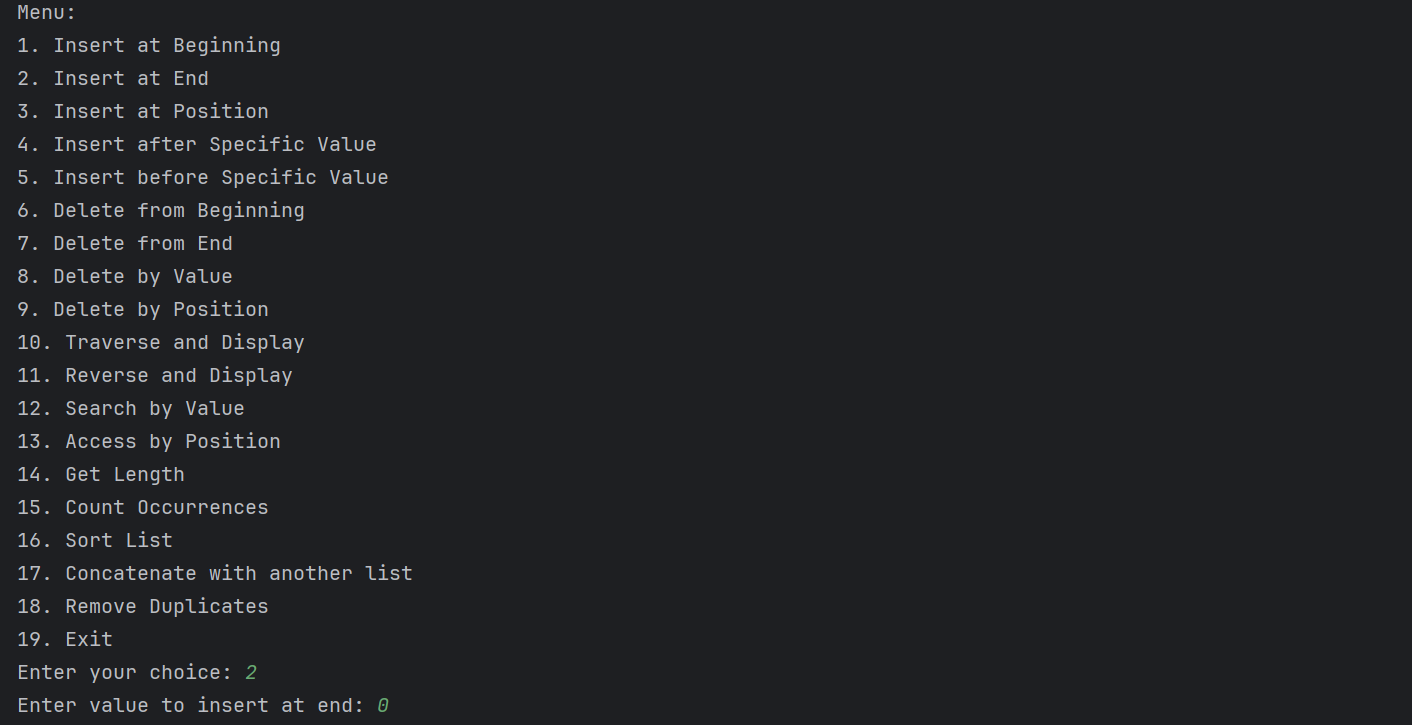
import java.util.Scanner;  
class Node {  
 int data;  
 Node next;  
 public Node(int data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
class LinkedList {  
 Node head;  
 public LinkedList() {  
 head = null;  
 }  
 public void insertAtBeginning(int data) {  
 Node newNode = new Node(data);  
 newNode.next = head;  
 head = newNode;  
 }  
 public void insertAtEnd(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode;  
 } else {  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = newNode;  
 }  
 }  
 public void insertAtPosition(int data, int position) {  
 if (position == 0) {  
 insertAtBeginning(data);  
 return;  
 }  
 Node newNode = new Node(data);  
 Node current = head;  
 for (int i = 0; i < position - 1 && current != null; i++) {  
 current = current.next;  
 }  
 if (current == null) {  
 System.*out*.println("Position out of bounds");  
 } else {  
 newNode.next = current.next;  
 current.next = newNode;  
 }  
 }  
 public void insertAfterValue(int data, int value) {  
 Node current = head;  
 while (current != null && current.data != value) {  
 current = current.next;  
 }  
 if (current != null) {  
 Node newNode = new Node(data);  
 newNode.next = current.next;  
 current.next = newNode;  
 } else {  
 System.*out*.println("Value not found");  
 }  
 }  
 public void insertBeforeValue(int data, int value) {  
 if (head == null) return;  
 if (head.data == value) {  
 insertAtBeginning(data);  
 return;  
 }  
 Node current = head;  
 while (current.next != null && current.next.data != value) {  
 current = current.next;  
 }  
 if (current.next != null) {  
 Node newNode = new Node(data);  
 newNode.next = current.next;  
 current.next = newNode;  
 } else {  
 System.*out*.println("Value not found");  
 }  
 }  
 public void deleteFromBeginning() {  
 if (head != null) {  
 head = head.next;  
 } else {  
 System.*out*.println("List is empty");  
 }  
 }  
 public void deleteFromEnd() {  
 if (head == null) {  
 System.*out*.println("List is empty");  
 return;  
 }  
 if (head.next == null) {  
 head = null;  
 return;  
 }  
 Node current = head;  
 while (current.next.next != null) {  
 current = current.next;  
 }  
 current.next = null;  
 }  
 public void deleteByValue(int value) {  
 if (head == null) return;  
 if (head.data == value) {  
 head = head.next;  
 return;  
 }  
 Node current = head;  
 while (current.next != null && current.next.data != value) {  
 current = current.next;  
 }  
 if (current.next != null) {  
 current.next = current.next.next;  
 } else {  
 System.*out*.println("Value not found");  
 }  
 }  
 public void deleteByPosition(int position) {  
 if (head == null) return;  
 if (position == 0) {  
 head = head.next;  
 return;  
 }  
 Node current = head;  
 for (int i = 0; i < position - 1 && current != null; i++) {  
 current = current.next;  
 }  
 if (current != null && current.next != null) {  
 current.next = current.next.next;  
 } else {  
 System.*out*.println("Position out of bounds");  
 }  
 }   
 public void traverse() {  
 Node current = head;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.println("null");  
 }  
 public void reverseTraversal(Node node) {  
 if (node == null) return;  
 reverseTraversal(node.next);  
 System.*out*.print(node.data + " -> ");  
 }  
 public void reversePrint() {  
 reverseTraversal(head);  
 System.*out*.println("null");  
 }   
 public void searchByValue(int value) {  
 Node current = head;  
 int position = 0;  
 while (current != null) {  
 if (current.data == value) {  
 System.*out*.println("Value " + value + " found at position " + position);  
 return;  
 }  
 current = current.next;  
 position++;  
 }  
 System.*out*.println("Value not found");  
 }  
 public void accessByPosition(int position) {  
 Node current = head;  
 for (int i = 0; i < position && current != null; i++) {  
 current = current.next;  
 }  
 if (current != null) {  
 System.*out*.println("Value at position " + position + " is " + current.data);  
 } else {  
 System.*out*.println("Position out of bounds");  
 }  
 }   
 public int getLength() {  
 Node current = head;  
 int length = 0;  
 while (current != null) {  
 length++;  
 current = current.next;  
 }  
 return length;  
 }  
 public int countOccurrences(int value) {  
 Node current = head;  
 int count = 0;  
 while (current != null) {  
 if (current.data == value) count++;  
 current = current.next;  
 }  
 return count;  
 }   
 public void sort() {  
 if (head == null || head.next == null) return;  
 Node current, index;  
 int temp;  
 for (current = head; current.next != null; current = current.next) {  
 for (index = current.next; index != null; index = index.next) {  
 if (current.data > index.data) {  
 temp = current.data;  
 current.data = index.data;  
 index.data = temp;  
 }  
 }  
 }  
 }   
 public void concatenate(LinkedList list2) {  
 if (head == null) {  
 head = list2.head;  
 return;  
 }  
 Node current = head;  
 while (current.next != null) {  
 current = current.next;  
 }  
 current.next = list2.head;  
 }   
 public void removeDuplicates() {  
 Node current = head;  
 while (current != null && current.next != null) {  
 Node index = current;  
 while (index.next != null) {  
 if (current.data == index.next.data) {  
 index.next = index.next.next;  
 } else {  
 index = index.next;  
 }  
 }  
 current = current.next;  
 }  
 }  
}  
public class ASSIGNMENT\_3\_1 {  
 public static void main(String[] args) {  
 LinkedList list = new LinkedList();  
 Scanner scanner = new Scanner(System.*in*);  
 boolean exit = false;   
 while (!exit) {  
 System.*out*.println("\nMenu:");  
 System.*out*.println("1. Insert at Beginning");  
 System.*out*.println("2. Insert at End");  
 System.*out*.println("3. Insert at Position");  
 System.*out*.println("4. Insert after Specific Value");  
 System.*out*.println("5. Insert before Specific Value");  
 System.*out*.println("6. Delete from Beginning");  
 System.*out*.println("7. Delete from End");  
 System.*out*.println("8. Delete by Value");  
 System.*out*.println("9. Delete by Position");  
 System.*out*.println("10. Traverse and Display");  
 System.*out*.println("11. Reverse and Display");  
 System.*out*.println("12. Search by Value");  
 System.*out*.println("13. Access by Position");  
 System.*out*.println("14. Get Length");  
 System.*out*.println("15. Count Occurrences");  
 System.*out*.println("16. Sort List");  
 System.*out*.println("17. Concatenate with another list");  
 System.*out*.println("18. Remove Duplicates");  
 System.*out*.println("19. Exit");  
 System.*out*.print("Enter your choice: ");  
 int choice = scanner.nextInt();  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter value to insert at beginning: ");  
 list.insertAtBeginning(scanner.nextInt());  
 break;  
 case 2:  
 System.*out*.print("Enter value to insert at end: ");  
 list.insertAtEnd(scanner.nextInt());  
 break;  
 case 3:  
 System.*out*.print("Enter value to insert: ");  
 int value = scanner.nextInt();  
 System.*out*.print("Enter position: ");  
 int position = scanner.nextInt();  
 list.insertAtPosition(value, position);  
 break;  
 case 4:  
 System.*out*.print("Enter value to insert: ");  
 int valueToInsert = scanner.nextInt();  
 System.*out*.print("Enter value after which to insert: ");  
 int afterValue = scanner.nextInt();  
 list.insertAfterValue(valueToInsert, afterValue);  
 break;  
 case 5:  
 System.*out*.print("Enter value to insert: ");  
 int beforeValue = scanner.nextInt();  
 System.*out*.print("Enter value before which to insert: ");  
 int insertBefore = scanner.nextInt();  
 list.insertBeforeValue(beforeValue, insertBefore);  
 break;  
 case 6:  
 list.deleteFromBeginning();  
 break;  
 case 7:  
 list.deleteFromEnd();  
 break;  
 case 8:  
 System.*out*.print("Enter value to delete: ");  
 int valueToDelete = scanner.nextInt();  
 list.deleteByValue(valueToDelete);  
 break;  
 case 9:  
 System.*out*.print("Enter position to delete: ");  
 int positionToDelete = scanner.nextInt();  
 list.deleteByPosition(positionToDelete);  
 break;  
 case 10:  
 System.*out*.println("Traversing the list:");  
 list.traverse();  
 break;  
 case 11:  
 System.*out*.println("Reverse traversing the list:");  
 list.reversePrint();  
 break;  
 case 12:  
 System.*out*.print("Enter value to search: ");  
 int valueToSearch = scanner.nextInt();  
 list.searchByValue(valueToSearch);  
 break;  
 case 13:  
 System.*out*.print("Enter position to access: ");  
 int positionToAccess = scanner.nextInt();  
 list.accessByPosition(positionToAccess);  
 break;  
 case 14:  
 System.*out*.println("Length of the list: " + list.getLength());  
 break;  
 case 15:  
 System.*out*.print("Enter value to count occurrences: ");  
 int valueToCount = scanner.nextInt();  
 int count = list.countOccurrences(valueToCount);  
 System.*out*.println("Occurrences of " + valueToCount + ": " + count);  
 break;  
 case 16:  
 System.*out*.println("Sorting the list:");  
 list.sort();  
 list.traverse();  
 break;  
 case 17:  
 LinkedList list2 = new LinkedList();  
 System.*out*.println("Enter elements for the second list, separated by space, end with -1:");  
 while (true) {  
 int element = scanner.nextInt();  
 if (element == -1) break;  
 list2.insertAtEnd(element);  
 }  
 list.concatenate(list2);  
 System.*out*.println("Concatenated list:");  
 list.traverse();  
 break;  
 case 18:  
 System.*out*.println("Removing duplicates from the list:");  
 list.removeDuplicates();  
 list.traverse();  
 break;  
 case 19:  
 exit = true;  
 break;  
 default:  
 System.*out*.println("Invalid choice! Please try again.");  
 }  
 }  
 scanner.close();  
 }  
}

**OUTPUT**

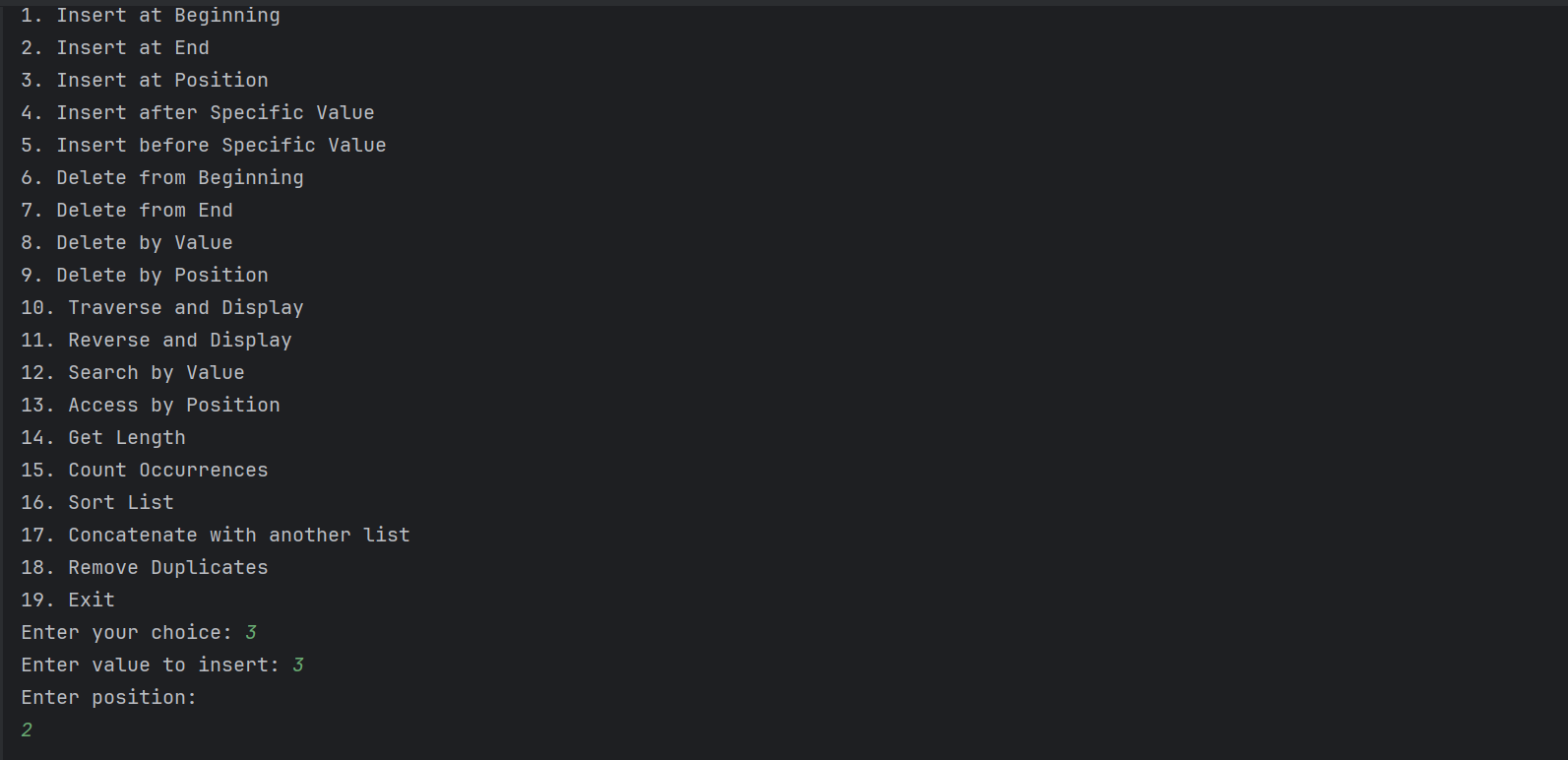
INSERTING ELEMENT AT BEGINNING



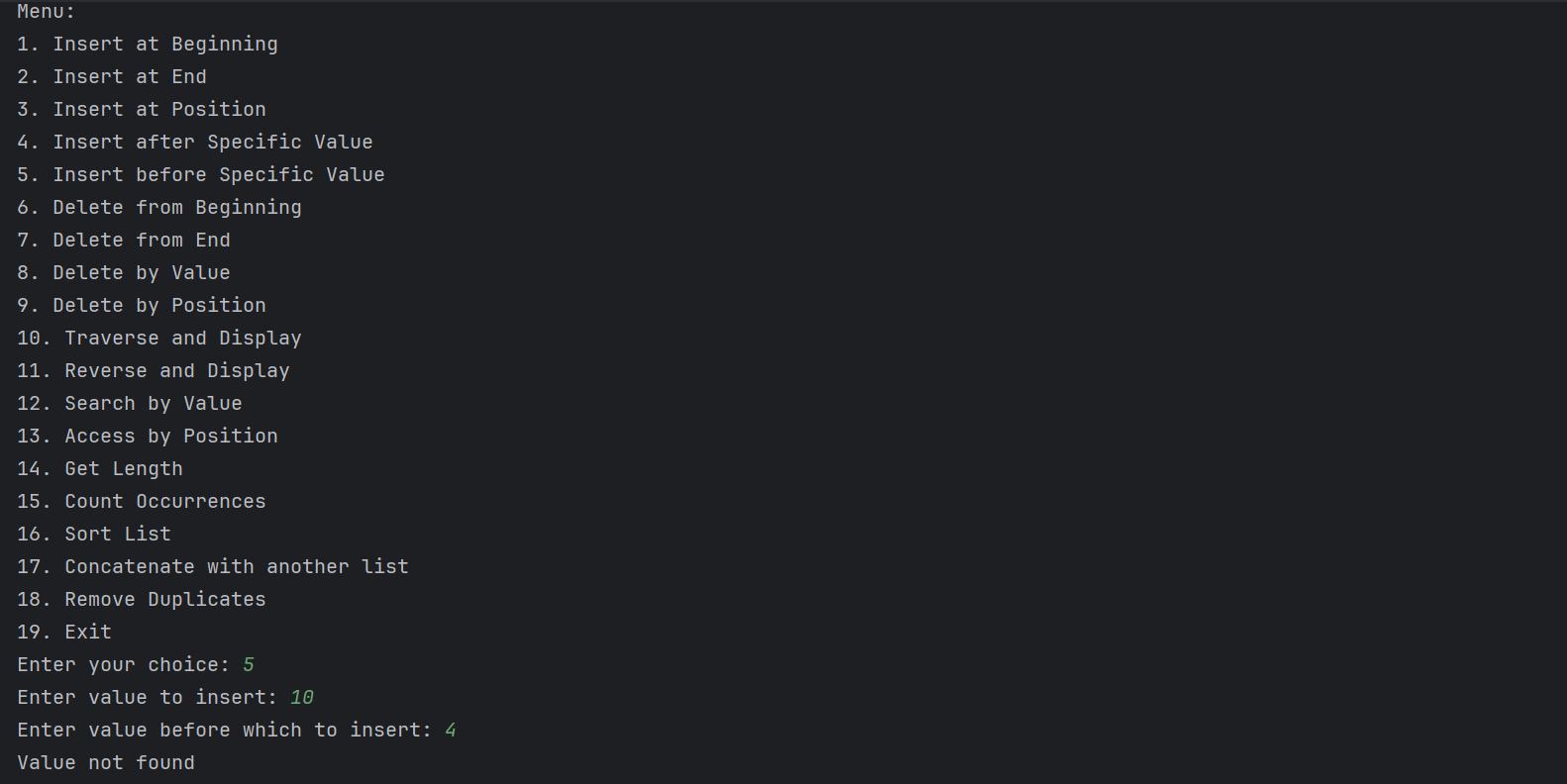
INSERT ELEMENT AT END



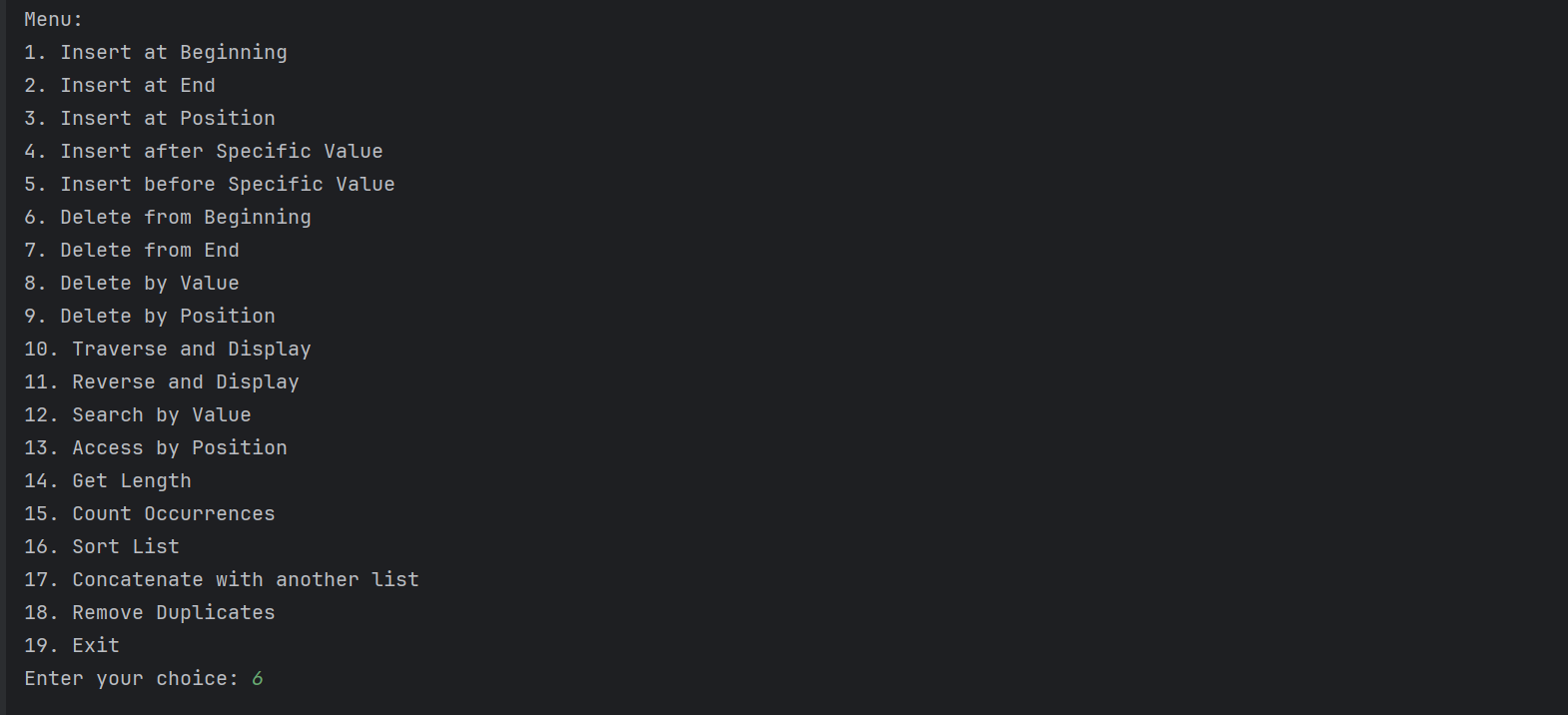
INSERT AT POSITION



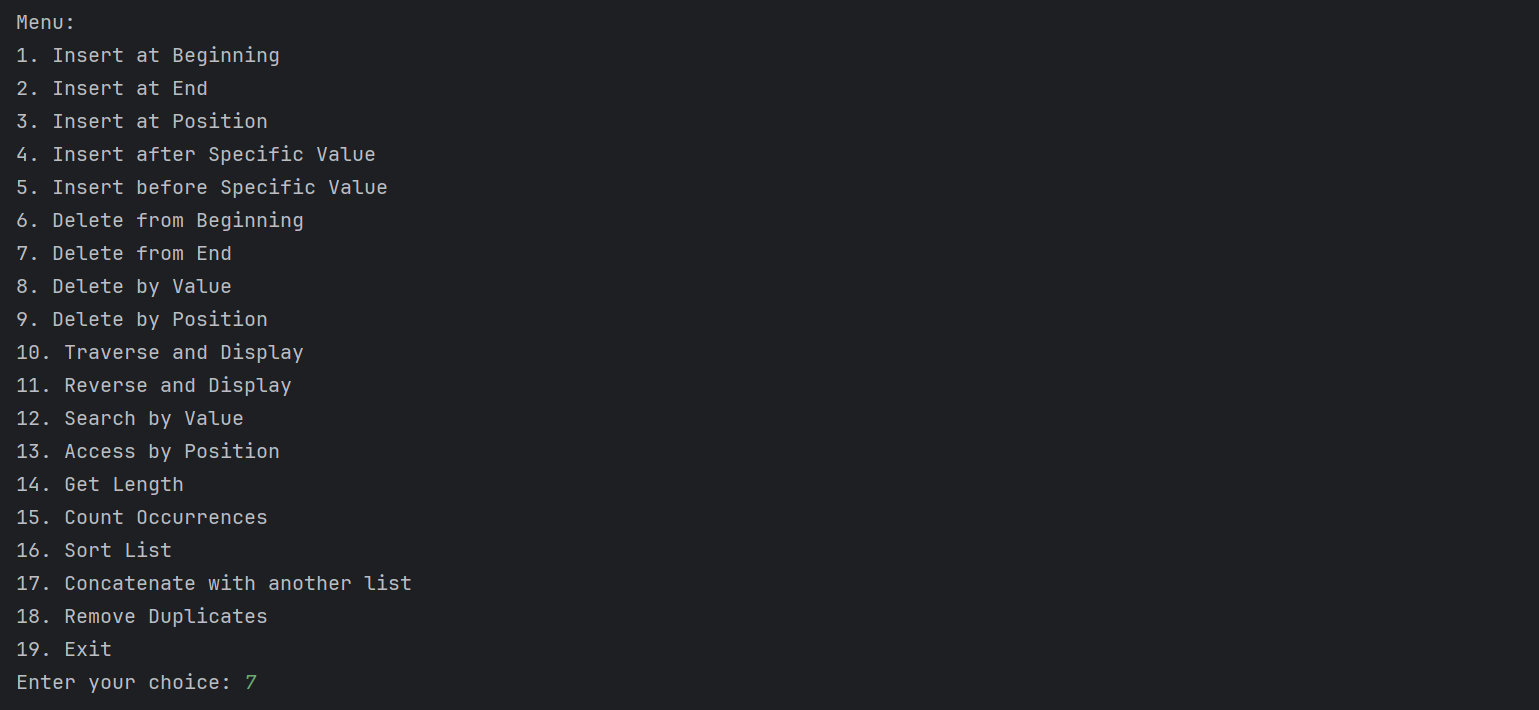
INSERT BEFORE SPECIFIED POSITION



DELETE FROM BEGINNING



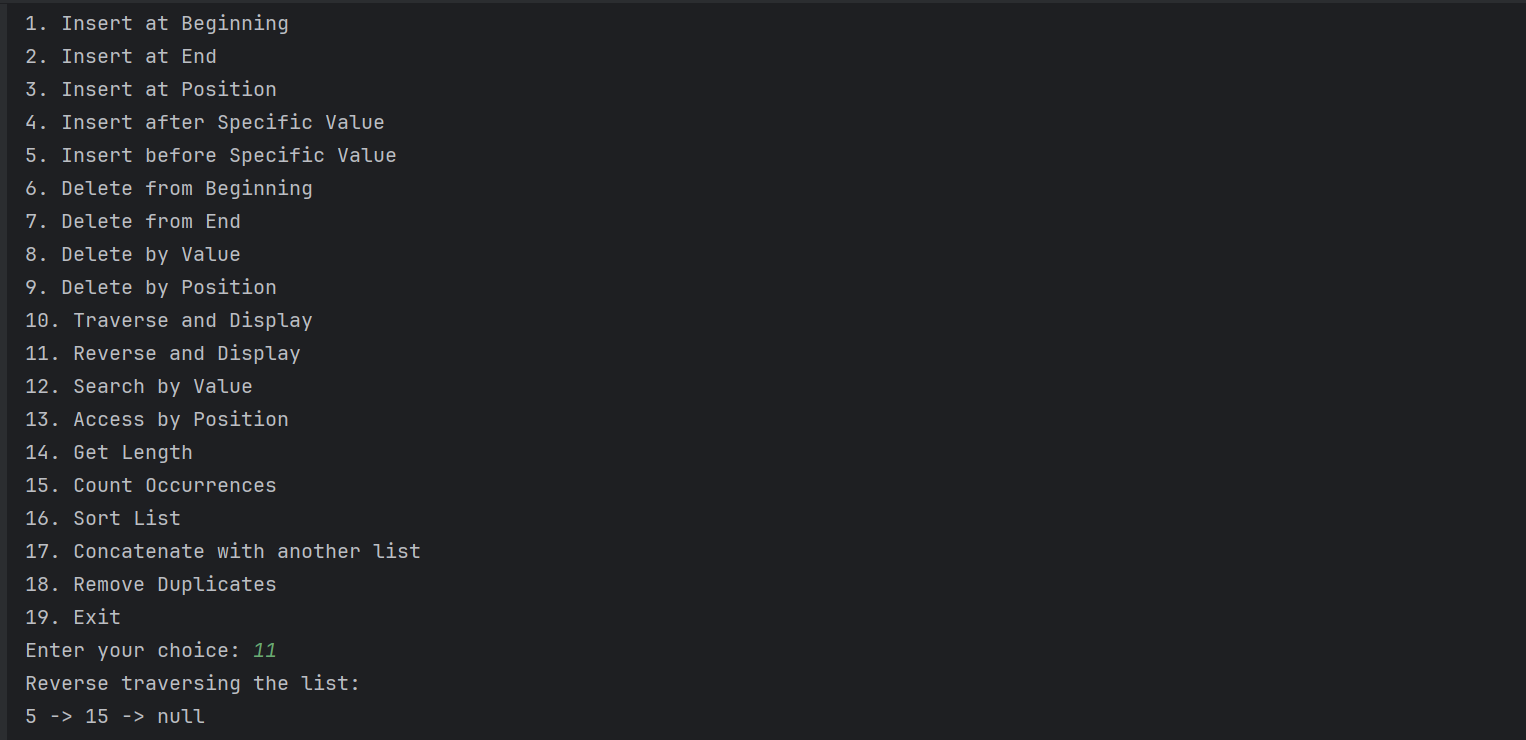
DELETE FROM END



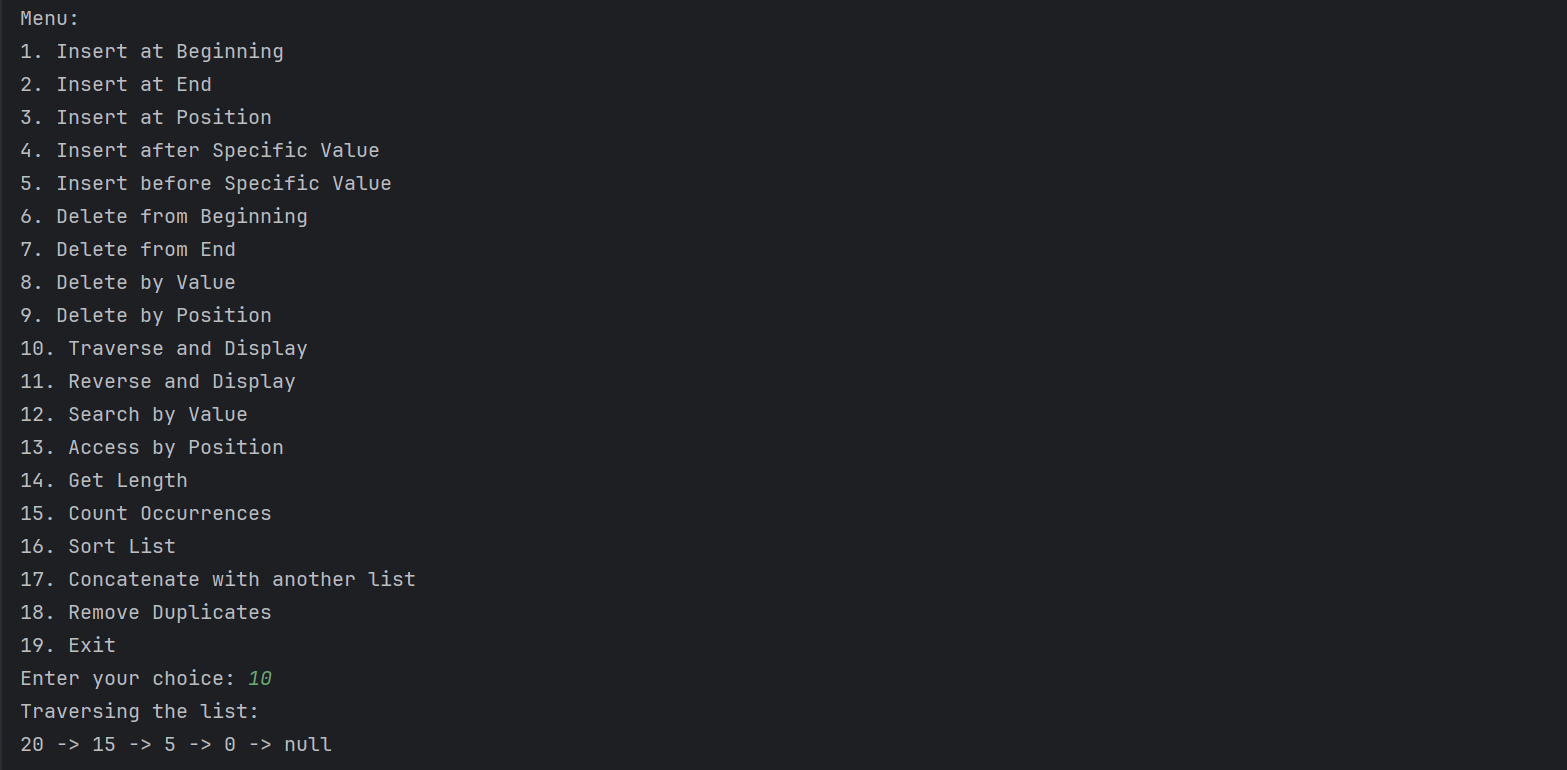
DELETE BY VALUE



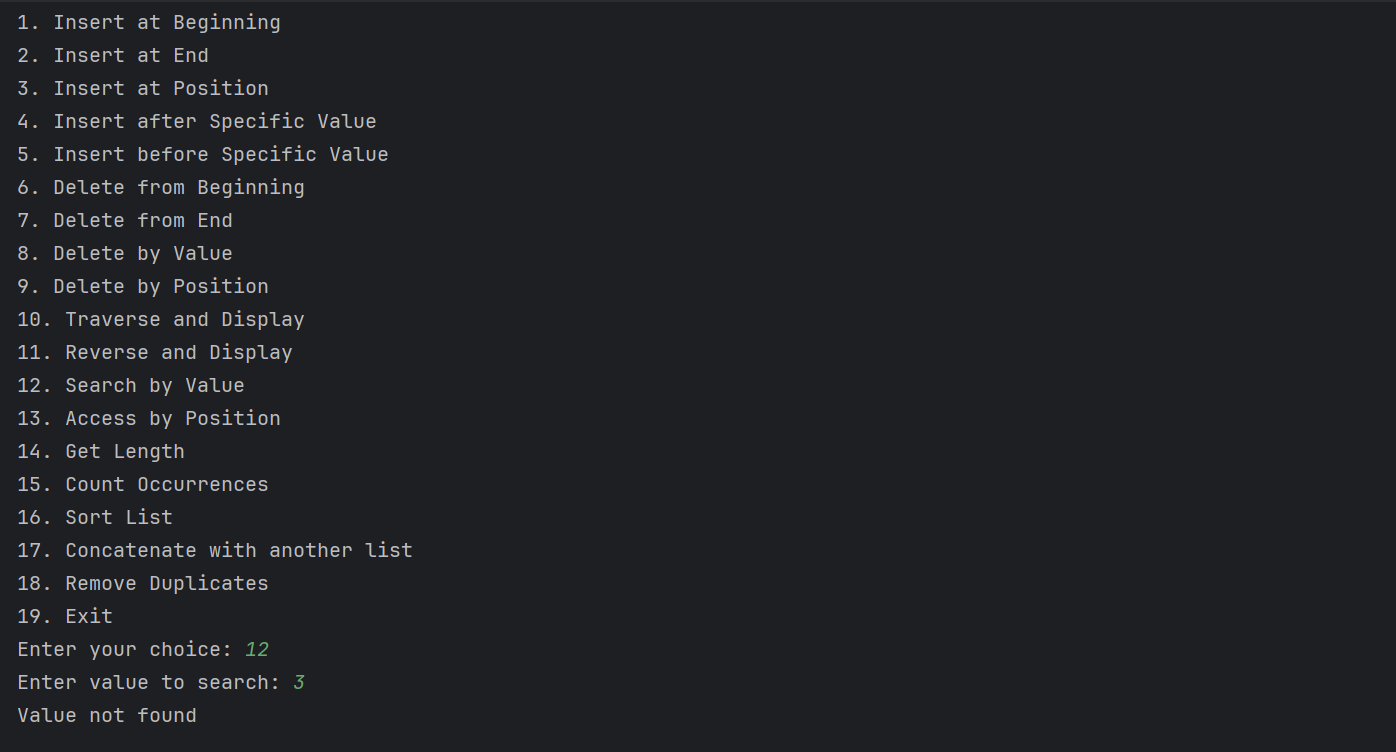
REVERSE TRAVERSING



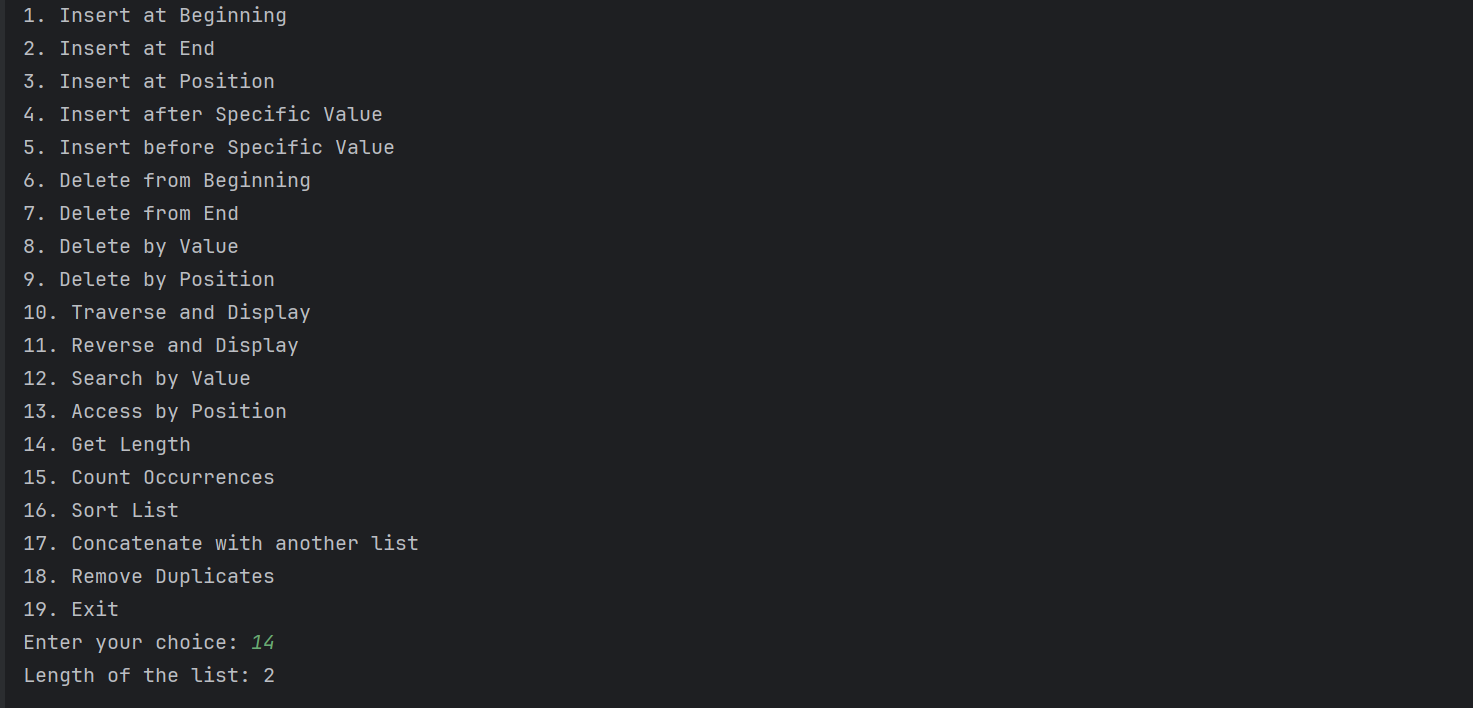
DISPLAYING ELEMENTS IN LINKED LIST



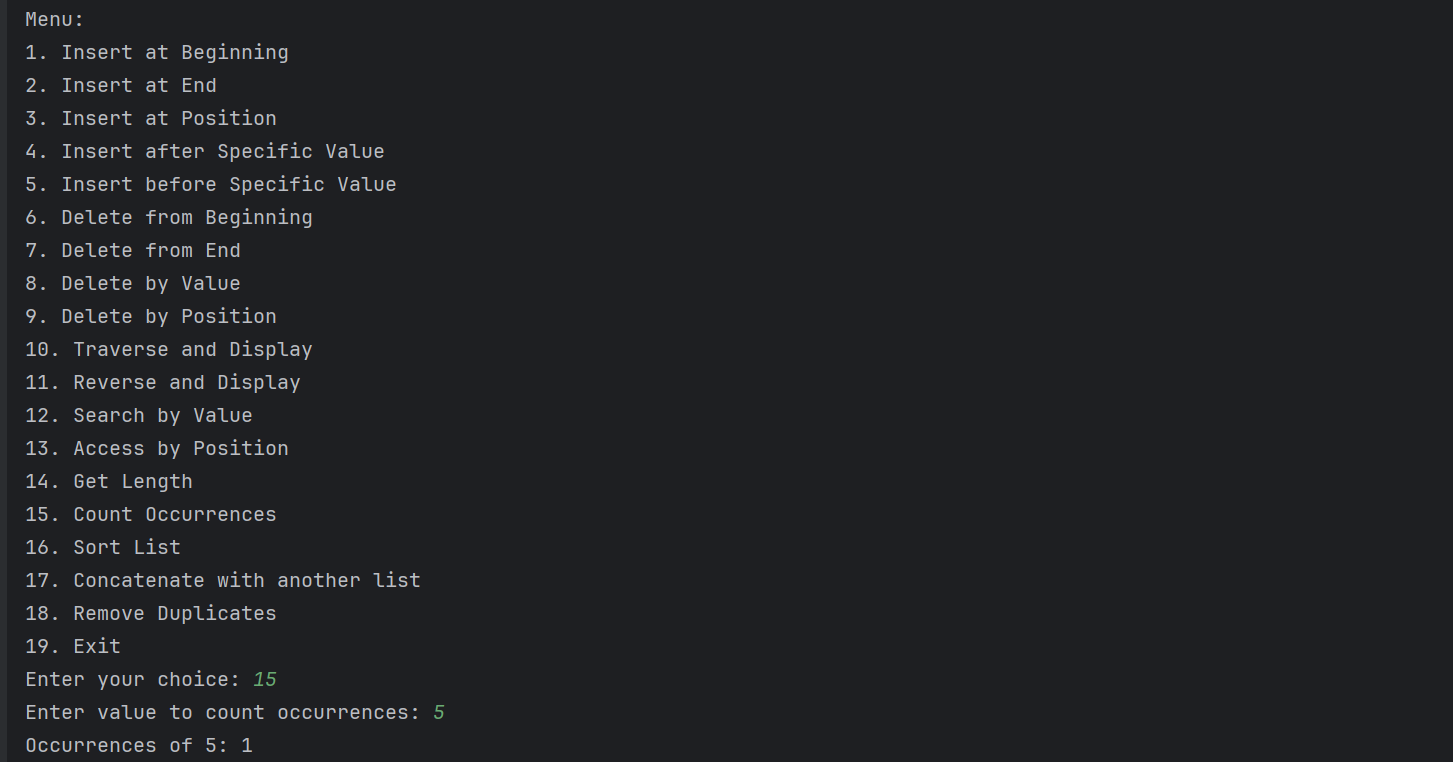
SEARCH BY VALUE



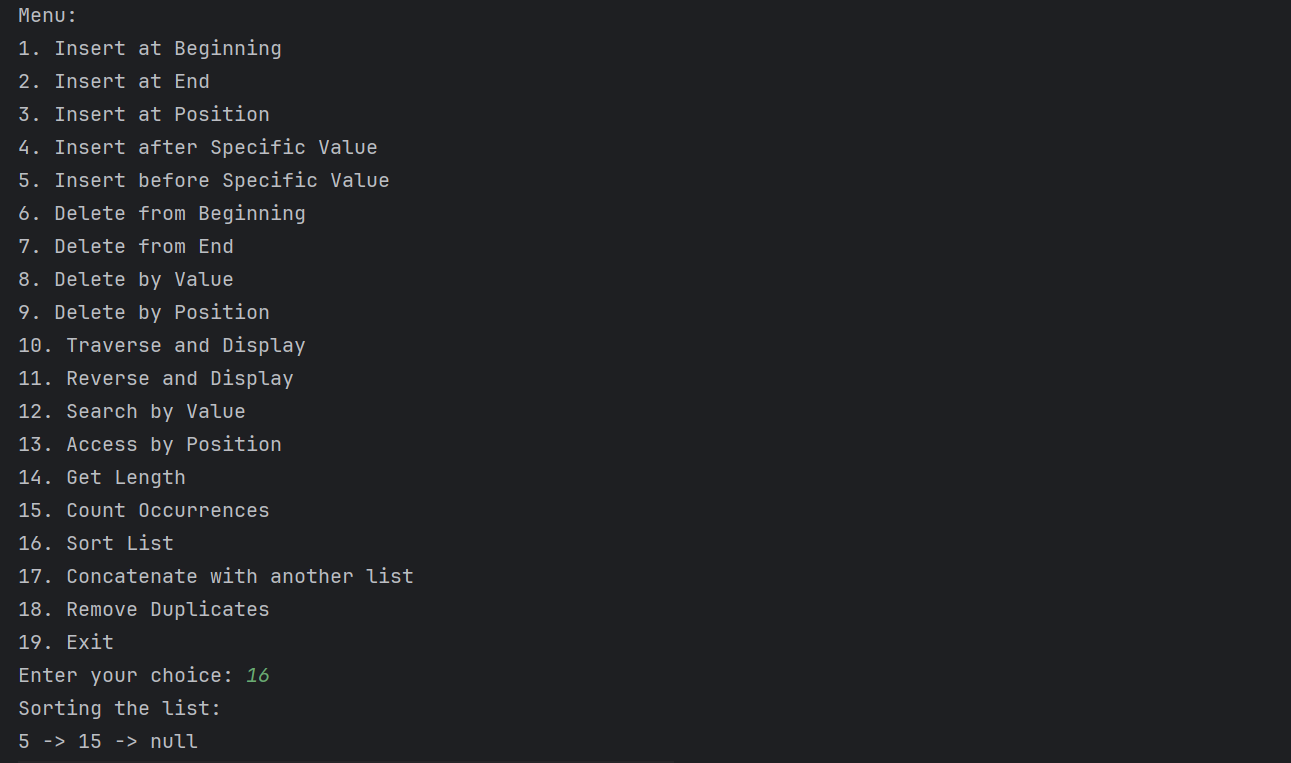
GET LENGTH



COUNT OCCURRENCE



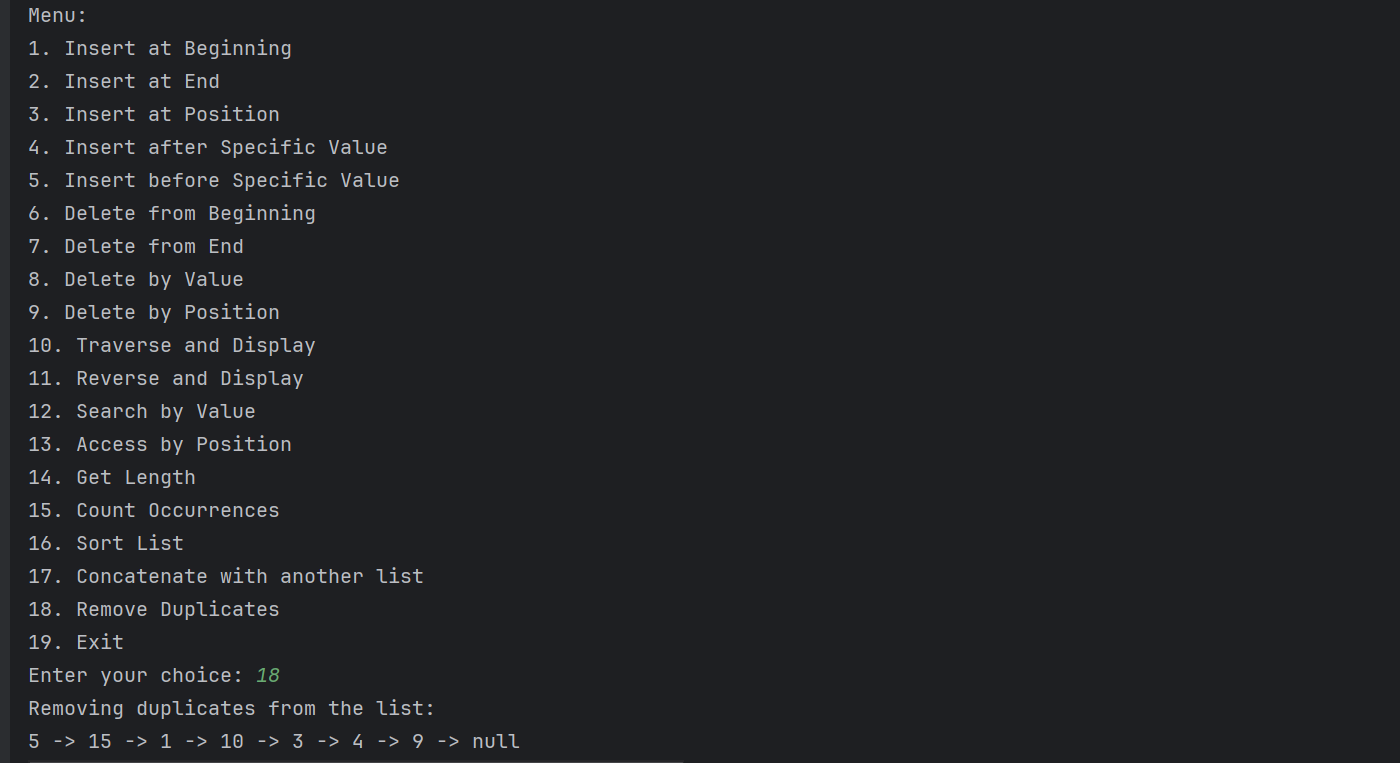
SORT LIST



CONCATENATION WITH ANOTHER LIST



REMOVE DUPLICATES



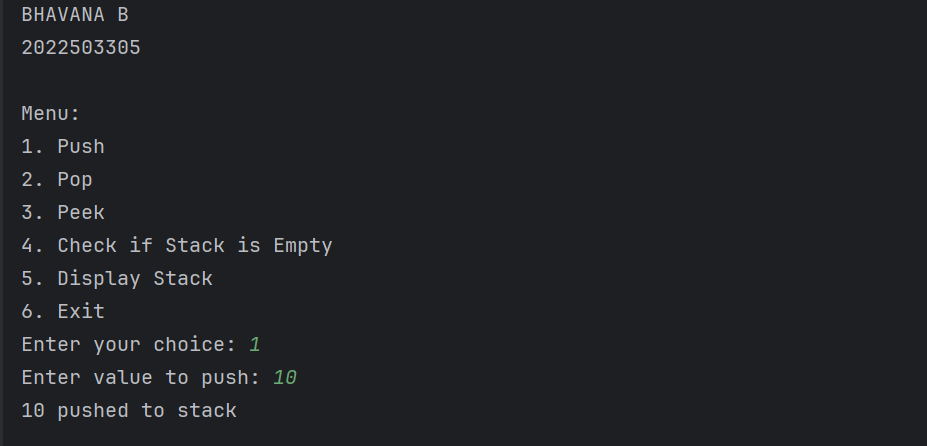
**EXERCISE 2: IMPLEMENTATION OF STACK USING LINKED LIST**

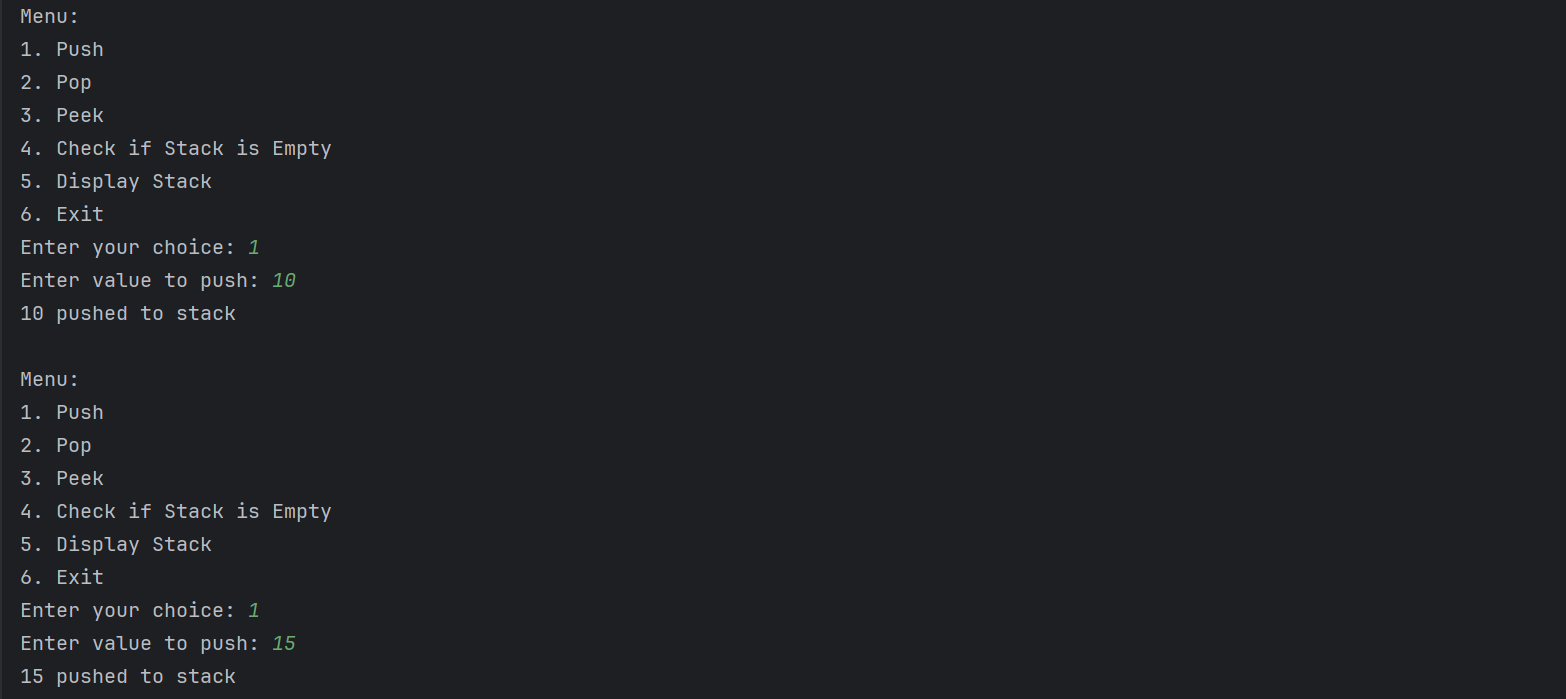
**PROGRAM**

import java.util.Scanner;  
class StackNode {   
 int data;  
 StackNode next;  
  
 public StackNode(int data) {  
 this.data = data;  
 this.next = null;  
 }  
}  
class Stack1 {  
 private StackNode top;  
 public Stack1() {  
 this.top = null;  
 }   
 public void push(int data) {  
 StackNode newNode = new StackNode(data);  
 newNode.next = top;  
 top = newNode;  
 System.*out*.println(data + " pushed to stack");  
 }   
 public int pop() {  
 if (top == null) {  
 System.*out*.println("Stack Underflow");  
 return -1;  
 } else {  
 int poppedData = top.data;  
 top = top.next;  
 System.*out*.println(poppedData + " popped from stack");  
 return poppedData;  
 }  
 }   
 public int peek() {  
 if (top == null) {  
 System.*out*.println("Stack is empty");  
 return -1;  
 } else {  
 System.*out*.println("Top element is " + top.data);  
 return top.data;  
 }  
 }   
 public boolean isEmpty() {  
 return top == null;  
 }   
 public void display() {  
 if (top == null) {  
 System.*out*.println("Stack is empty");  
 } else {  
 StackNode current = top;  
 while (current != null) {  
 System.*out*.print(current.data + " -> ");  
 current = current.next;  
 }  
 System.*out*.println("null");  
 }  
 }  
}  
public class ASSIGNMENT\_3\_2 {  
 public static void main(String[] args) {  
 Stack stack = new Stack();  
 Scanner scanner = new Scanner(System.*in*);  
 boolean exit = false;  
  
 while (!exit) {  
 System.*out*.println("\nMenu:");  
 System.*out*.println("1. Push");  
 System.*out*.println("2. Pop");  
 System.*out*.println("3. Peek");  
 System.*out*.println("4. Check if Stack is Empty");  
 System.*out*.println("5. Display Stack");  
 System.*out*.println("6. Exit");  
 System.*out*.print("Enter your choice: ");  
 int choice = scanner.nextInt();  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter value to push: ");  
 int value = scanner.nextInt();  
 stack.push(value);  
 break;  
 case 2:  
 stack.pop();  
 break;  
 case 3:  
 stack.peek();  
 break;  
 case 4:  
 if (stack.isEmpty()) {  
 System.*out*.println("Stack is empty");  
 } else {  
 System.*out*.println("Stack is not empty");  
 }  
 break;  
 case 5:  
 stack.display();  
 break;  
 case 6:  
 exit = true;  
 break;  
 default:  
 System.*out*.println("Invalid choice! Please try again.");  
 }  
 }  
  
 scanner.close();  
 }  
}

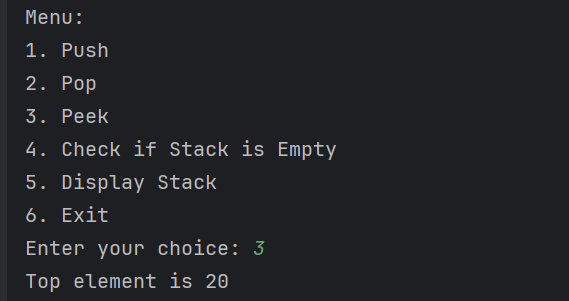
OUTPUT

PUSH OPERATION

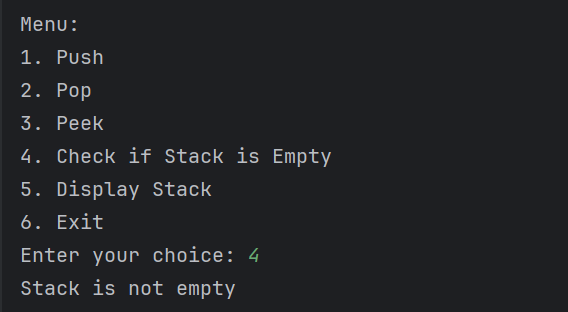




PEEK OPERATION



CHECK IF STACK EMPTY



DISPLAY STACK

