Task 1: Single Linked List and Palindrom Function

ANSWER

```
#include <iostream>
using namespace std;
class Node {
private:
   int data;
   Node* next;

public:
   Node* head;

   Node() {
      head = NULL;
   }

   void SLL(int n) {

      if (head == NULL) {

         head = new Node();
         head->data = n;
         head->next = NULL;

      }

      else {

         Node* p;
         p = new Node();
         p->data = n;
         p->next = head;
         head = p;

      }
   }

   void display() {
      Node* ptr;
```

```
    ptr = head;
    if (ptr == NULL) {
        cout << " \nNo data is in the list.." << endl;
        return;
    }
    else {

        while (ptr != NULL) {
            cout << ptr->data << endl;
            ptr = ptr->next;
        }
    }

}

bool isPalindrome() {
    if (head == NULL) {
        cout << "The list is empty, so it's considered a palindrome." << endl;
        return true;
    }

    Node* fast = head;
    Node* slow = head;
    Node* prev_slow = head;
    Node* mid = NULL;
    bool isPalindrome = true;

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

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

    Node* second_half = slow;
    prev_slow->next = NULL;
```

```
    reverseList(head);

    isPalindrome = compareLists(head, second_half);

    reverseList(second_half);

    if (mid != NULL) {
        prev_slow->next = mid;
        mid->next = second_half;
    } else {
        prev_slow->next = second_half;
    }

    return isPalindrome;
}

void reverseList(Node*& head) {
    Node* prev = NULL;
    Node* current = head;
    Node* next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    head = prev;
}

bool compareLists(Node* head1, Node* head2) {
    Node* temp1 = head1;
    Node* temp2 = head2;

    while (temp1 && temp2) {
        if (temp1->data == temp2->data) {
            temp1 = temp1->next;
            temp2 = temp2->next;
        } else {
```

```
            return false;
        }
    }

    if (!temp1 && !temp2) {
        return true;
    }
    return false;
  }
};

int main() {
  Node n;
  n.SLL(1);
  n.SLL(2);
  n.SLL(2);
  n.SLL(1);
  n.display();

  if (n.isPalindrome()) {
    cout << "The linked list is a palindrome." << endl;
  } else {
    cout << "The linked list is not a palindrome." << endl;
  }

  return 0;
}
```

OUTPUT

Task 2: Conditional Branching
Implement Stack using Array
ANSWER

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

const int SIZE = 100;

class Stack{
private:
   int top;
   int arr[SIZE];

public:
   Stack() {
      top = -1;
   }
   bool isEmpty() {
      return top == -1;
   }
   bool isFull() {
      return top == SIZE - 1;
   }
   void push(int data) {
      if (isFull()) {
         cout << "Stack is full!!!!!" << endl;
         return;
      }
      arr[++top] = data;
   }
   void pop() {
      if (isEmpty()) {
         cout << "Stack is empty!!!!!" << endl;
         return;
      }
      --top;
   }
   int peek() {
```

```
        if (isEmpty()) {
            cout << "Stack is empty!!!!!" << endl;
            return -1;
        }
        return arr[top];
    }
};

int main() {
    Stack stack;

    cout << "Select an Operation:" << endl;
    cout << "1. to Push" << endl;
    cout << "2. to Pop" << endl;
    cout << "3. to Peek" << endl;
    cout << "4. Is stack Full" << endl;
    cout << "5. Is stack Empty" << endl;
    cout << "6. Quit" << endl;

    int c, d;

    do {
        cout << "Choose an Option : ";
        cin >> c;

        switch (c) {
            case 1:
                cout << "Enter data: ";
                cin >> d;
                stack.push(d);
                break;
            case 2:
                stack.pop();
                break;
            case 3:
                cout << "Top element of stack is " << stack.peek() << endl;
                break;
            case 4:
                if (stack.isFull()) {
                    cout << "Stack is full." << endl;
```

```
            } else {
                cout << "Stack is not full." << endl;
            }
            break;
        case 5:
            if (stack.isEmpty()) {
                cout << "Stack is empty." << endl;
            } else {
                cout << "Stack is not empty." << endl;
            }
            break;
        case 6:
            cout << "Exiting program....." << endl;
            break;
        default:
            cout << "Select a valid operation" << endl;
      }
   } while (choice != 6);

   return 0;
}
```

OUTPUT