PROBLEM SOLVING PROGRAMMING ASSIGNMENT I

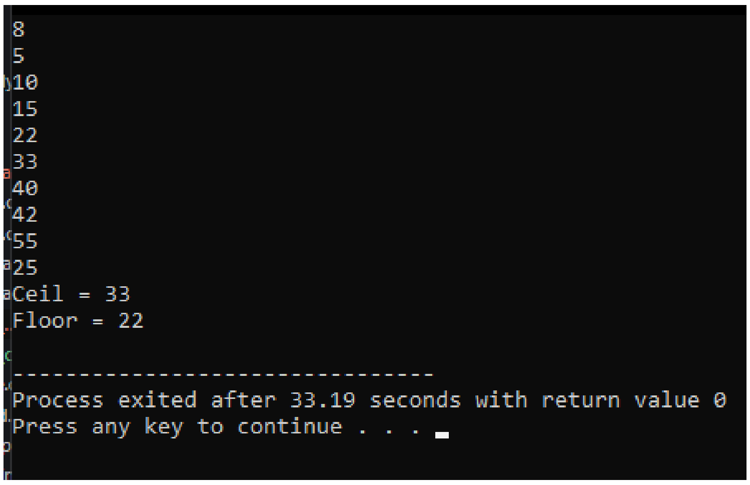
Natya Vidhan Biswas

B.Sc. CS. Hons. 25771

Q1. Give the ceiling and floor of any input integer in an integer array

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

void findFloorCeil(int arr[], int n, int d, int &ceilVal,
int &floorVal) {
    int low = 0, high = n - 1;
    floorVal = -1;
    ceilVal = -1;

    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == d) {
            ceilVal = floorVal = d;
            return;
        } else if (arr[mid] < d) {
            floorVal = arr[mid];
            low = mid + 1;
        } else {
            ceilVal = arr[mid];
            high = mid - 1;
        }
    }
}

int main() {
    int n;
    cin >> n;

    int* arr = new int[n];
    for (int i = 0; i < n; i++) cin >> arr[i];

    int d;
    cin >> d;
```

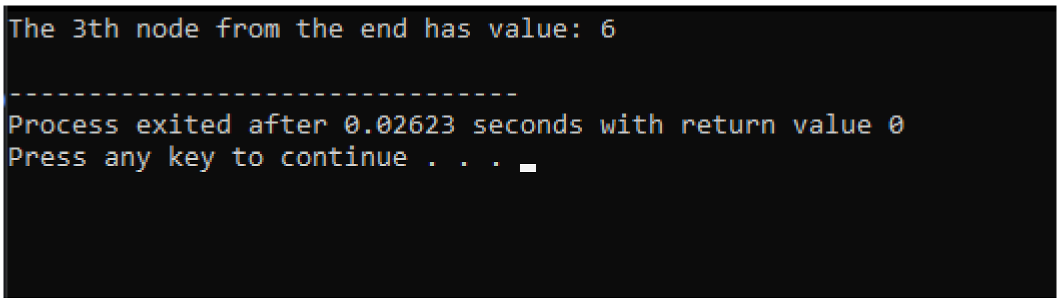
```cpp
    int ceilVal, floorVal;
    findFloorCeil(arr, n, d, ceilVal, floorVal);

    cout << "Ceil = " << ceilVal << endl;
    cout << "Floor = " << floorVal << endl;

    delete[] arr;
    return 0;
}
```



```
8
5
10
15
22
33
40
42
55
25
Ceil = 33
Floor = 22

--------------------------------
Process exited after 33.19 seconds with return value 0
Press any key to continue . . .
```

Q2. Given a singly linked list of integers, write an iterative function that returns the address of the ith node from the end of the list. Also, make sure to not use size data member directly or indirectly (by calculating size via making a traversal). i is a 0-based index. Assume that valid values of k will be passed.

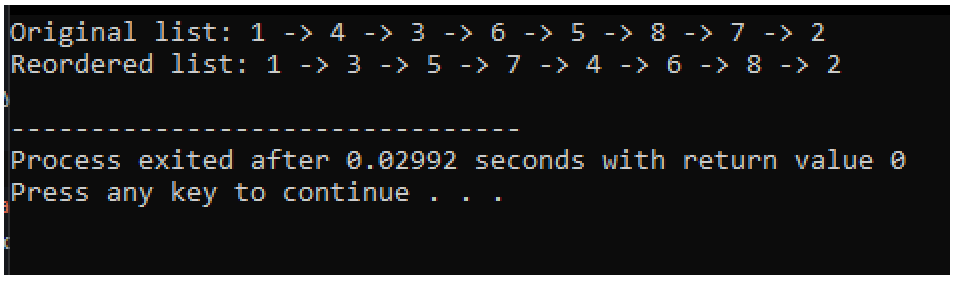
```
#include <iostream>
using namespace std;

class Node {
    int value;
    Node *next;
public:
    Node(int val=0, Node *n=0) {
        value = val;
        next = n;
    }
    int getValue() { return value; }
    friend class LinkedList;
};

class LinkedList {
    Node *head;
public:
    LinkedList() {
        head = 0;
    }

    void add_end(int val) {
        Node *p;
        p = new Node(val);
        if (head == 0) {
            head = p;
            return;
        }
        Node *q;
        q = head;
        while (q->next)
        {
            q = q->next;
        }
        q->next = p;
        return;
    }

    Node* get_ith_from_end(int i) {
        Node *main_ptr = head;
        Node *ref_ptr = head;

        int count = 0;
        if (head != 0) {
            while (count < i) {
                if (ref_ptr == 0) {
                    cout << i << " is greater than the 
number of nodes in list\n";
                    return 0;
                }
                ref_ptr = ref_ptr->next;
                count++;
            }

            while (ref_ptr != 0) {
                main_ptr = main_ptr->next;
                ref_ptr = ref_ptr->next;
            }
        }
        return main_ptr;
    }
};

int main() {
    LinkedList list;
    int nums[10] = {1, 3, 5, 7, 9, 2, 4, 6, 8, 0};
    for (int i = 0; i < 10; i++) {
        list.add_end(nums[i]);
    }

    int i = 3;
    Node* node = list.get_ith_from_end(i);
    if (node)
        cout << "The " << i << "th node from the end has 
value: " << node->getValue() << endl;
    else
        cout << "Index out of bounds." << endl;

    return 0;
}
```



```
The 3th node from the end has value: 6

--------------------------------
Process exited after 0.02623 seconds with return value 0
Press any key to continue . . .
```

Q3. Given a singly linked list of odd and even integers, write a function that reorders the list such all odd values are followed by all even values. The relative order of elements should not change. Also, take care of the cases when there are no odd or no even elements.

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

class Node {
    int value;
    Node *next;
public:
    Node(int val=0, Node *n=0) {
        value = val;
        next = n;
    }
    int getValue() { return value; }
    friend class LinkedList;
};

class LinkedList {
    Node *head;
public:
    LinkedList() {
        head = 0;
    }

    void add_end(int val) {
        Node *p;
        p = new Node(val);
        if (head == 0) {
            head = p;
            return;
        }
        Node *q;
        q = head;
        while (q->next)
        {
            q = q->next;
        }
        q->next = p;
        return;
    }
        void reorder_odd_even() {
        if (!head || !head->next) return;

        Node* oddHead = 0;
        Node* oddTail = 0;
        Node* evenHead = 0;
        Node* evenTail = 0;

        Node* current = head;

        while (current) {
            Node* nextNode = current->next;
            current->next = 0;

            if (current->value % 2 != 0) {
                if (!oddHead) {
                    oddHead = oddTail = current;
                } else {
                    oddTail->next = current;
                    oddTail = current;
                }
            } else {
                if (!evenHead) {
                    evenHead = evenTail = current;
                } else {
                    evenTail->next = current;
                    evenTail = current;
                }
            }

            current = nextNode;
        }
        if (oddTail) {
            oddTail->next = evenHead;
            head = oddHead;
        } else {
            head = evenHead;
        }
    }
    void print() {
        Node *p;
        p = head;
        while (p)
        {
            cout << p->value;
            if (p->next) {
                cout << " -> ";
            }
            p = p->next;
        }
        cout << "\n";
        return;
    }
};

int main() {
    LinkedList list;
    int nums[8] = {1, 4, 3, 6, 5, 8, 7, 2};
    for (int i = 0; i < 8; i++) {
        list.add_end(nums[i]);
    }

    cout << "Original list: ";
    list.print();

    list.reorder_odd_even();

    cout << "Reordered list: ";
    list.print();
    return 0;
}
```

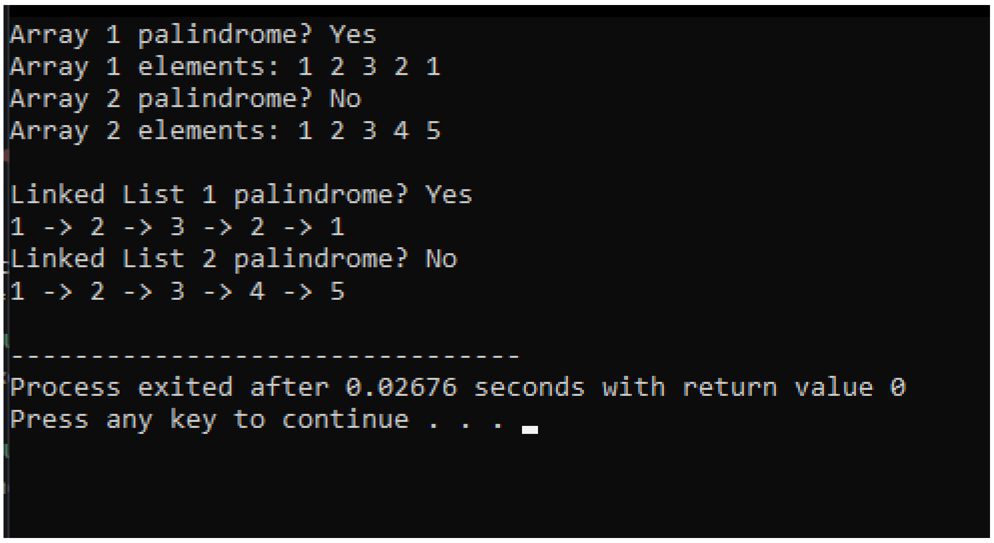


```
Original list: 1 -> 4 -> 3 -> 6 -> 5 -> 8 -> 7 -> 2
Reordered list: 1 -> 3 -> 5 -> 7 -> 4 -> 6 -> 8 -> 2

--------------------------------
Process exited after 0.02992 seconds with return value 0
Press any key to continue . . .
```

Q4. Check a given sequence of elements is a palindrome using an array as well as a linked list.

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

class Node {
public:
    int value;
    Node* next;
    Node(int val = 0, Node* n = 0) {
        value = val;
        next = n;
    }
};

class LinkedList {
    Node* head;
public:
    LinkedList() { head = 0; }

    void add_end(int val) {
        Node* p = new Node(val);
        if (!head) {
            head = p;
            return;
        }
        Node* q = head;
        while (q->next) q = q->next;
        q->next = p;
    }

    void print() {
        Node* p = head;
        while (p) {
            cout << p->value;
            if (p->next) cout << " -> ";
            p = p->next;
        }
        cout << "\n";
    }

    Node* reverseList(Node* node) {
        Node* prev = 0;
        Node* curr = node;
        while (curr) {
            Node* nextNode = curr->next;
            curr->next = prev;
            prev = curr;
            curr = nextNode;
        }
        return prev;
    }

    bool isPalindrome() {
        if (!head || !head->next) return true;

        Node* slow = head;
        Node* fast = head;

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

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

        Node* p1 = head;
        Node* p2 = secondHalf;
        bool result = true;
        while (p2) {
            if (p1->value != p2->value) {
                result = false;
                break;
            }
            p1 = p1->next;
```

```
            p2 = p2->next;
        }

        slow->next = reverseList(secondHalf);

        return result;
    }
};

bool isPalindromeArray(int arr[], int n) {
    int left = 0, right = n - 1;
    while (left < right) {
        if (arr[left] != arr[right])
            return false;
        left++;
        right--;
    }
    return true;
}

int main() {

    int nums[] = {1, 2, 3, 2, 1};
    int nums2[] = {1, 2, 3, 4, 5};
    int n = 5;

    // ----- Array Palindrome -----
    cout << "Array 1 palindrome? " <<
(isPalindromeArray(nums, n) ? "Yes" : "No") << "\n";
    cout << "Array 1 elements: ";
    for (int i = 0; i < n; i++) {
        cout << nums[i] << " ";
    }
    cout << "\n";
    cout << "Array 2 palindrome? " <<
(isPalindromeArray(nums2, n) ? "Yes" : "No") << "\n";
    cout << "Array 2 elements: ";
    for (int i = 0; i < n; i++) {
        cout << nums2[i] << " ";
    }
    cout << "\n\n";

    // ----- Linked List Palindrome -----
    LinkedList list;
    for (int i = 0; i < n; i++) {
        list.add_end(nums[i]);
    }
    LinkedList list2;
    for (int i = 0; i < n; i++) {
        list2.add_end(nums2[i]);
    }

    cout << "Linked List 1 palindrome? " <<
(list.isPalindrome() ? "Yes" : "No") << "\n";
    list.print();
    cout << "Linked List 2 palindrome? " <<
(list2.isPalindrome() ? "Yes" : "No") << "\n";
    list2.print();

    return 0;
}
```

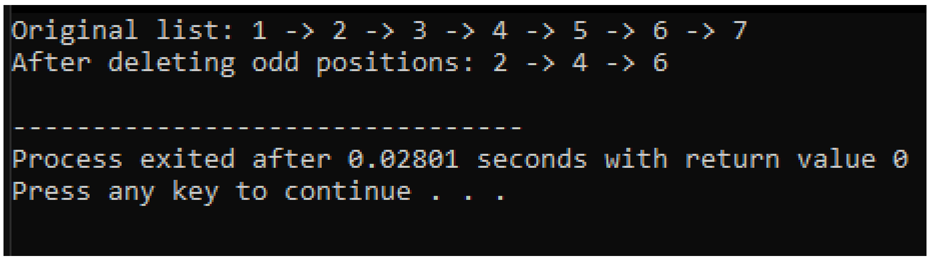


```
Array 1 palindrome? Yes
Array 1 elements: 1 2 3 2 1
Array 2 palindrome? No
Array 2 elements: 1 2 3 4 5

Linked List 1 palindrome? Yes
1 -> 2 -> 3 -> 2 -> 1
Linked List 2 palindrome? No
1 -> 2 -> 3 -> 4 -> 5

--------------------------------
Process exited after 0.02676 seconds with return value 0
Press any key to continue . . .
```

Q5. Delete all odd position nodes from a given linked list.

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

class Node {
public:
    int value;
    Node* next;
    Node(int val = 0, Node* n = 0) {
        value = val;
        next = n;
    }
};

class LinkedList {
    Node* head;
public:
    LinkedList() { head = 0; }

    void add_end(int val) {
        Node* p = new Node(val);
        if (!head) {
            head = p;
            return;
        }
        Node* q = head;
        while (q->next) q = q->next;
        q->next = p;
    }

    void print() {
        Node* p = head;
        while (p) {
            cout << p->value;
            if (p->next) cout << " -> ";
            p = p->next;
        }
        cout << "\n";
    }
    void delete_odd_positions() {
        if (!head) return;

        Node* current = head;
        Node* prev = 0;
        int pos = 1;

        while (current) {
            Node* nextNode = current->next;

            if (pos % 2 != 0) {
                if (prev) {
                    prev->next = nextNode;
                } else {
                    head = nextNode;
                }
                delete current;
            } else {
                prev = current;
            }

            current = nextNode;
            pos++;
        }
    }
};


int main() {
    LinkedList list;
    int nums[] = {1, 2, 3, 4, 5, 6, 7};
    for (int i = 0; i < 7; i++) {
        list.add_end(nums[i]);
    }

    cout << "Original list: ";
    list.print();
```

```cpp
    list.delete_odd_positions();

    cout << "After deleting odd positions: ";
    list.print();

    return 0;
}
```

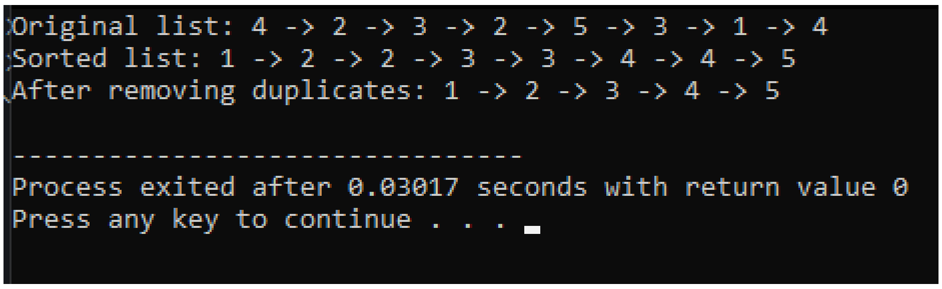


```
Original list: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7
After deleting odd positions: 2 -> 4 -> 6

--------------------------------
Process exited after 0.02801 seconds with return value 0
Press any key to continue . . .
```

Q6. Remove duplicates from a linked list

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

class Node {
    int value;
    Node *next;
public:
    Node(int val=0, Node *n=0) {
        value = val;
        next = n;
    }
    int getValue() { return value; }
    friend class LinkedList;
};

class LinkedList {
    Node* head;
public:
    LinkedList() { head = 0; }

    void add_end(int val) {
        Node* p = new Node(val);
        if (!head) { head = p; return; }
        Node* q = head;
        while (q->next) q = q->next;
        q->next = p;
    }

    void print() {
        Node* p = head;
        while (p) {
            cout << p->value;
            if (p->next) cout << " -> ";
            p = p->next;
        }
        cout << "\n";
    }
    void bubbleSort() {
        if (!head || !head->next) return;

        bool swapped;
        Node* ptr1;
        Node* lptr = 0;

        do {
            swapped = false;
            ptr1 = head;

            while (ptr1->next != lptr) {
                if (ptr1->value > ptr1->next->value) {
                    swap(ptr1->value, ptr1->next->value);
                    swapped = true;
                }
                ptr1 = ptr1->next;
            }
            lptr = ptr1;
        } while (swapped);
    }

    void removeDuplicatesSorted() {
        Node* current = head;
        while (current && current->next) {
            if (current->value == current->next->value) {
                Node* temp = current->next;
                current->next = temp->next;
                delete temp;
            } else {
                current = current->next;
            }
        }
    }
};

int main() {
```

```cpp
    LinkedList list;
    int nums[] = {4, 2, 3, 2, 5, 3, 1, 4};
    for (int i = 0; i < 8; i++) {
        list.add_end(nums[i]);
    }

    cout << "Original list: ";
    list.print();

    list.bubbleSort();
    cout << "Sorted list: ";
    list.print();

    list.removeDuplicatesSorted();
    cout << "After removing duplicates: ";
    list.print();

    return 0;
}
```



```
Original list: 4 -> 2 -> 3 -> 2 -> 5 -> 3 -> 1 -> 4
Sorted list: 1 -> 2 -> 2 -> 3 -> 3 -> 4 -> 4 -> 5
After removing duplicates: 1 -> 2 -> 3 -> 4 -> 5

--------------------------------
Process exited after 0.03017 seconds with return value 0
Press any key to continue . . .
```