Problem - A :

Code : #include <iostream>
using namespace std;

```
struct Node {
    int data;
    Node* prev;
    Node* next;
};

Node* newNode(int val) {
    Node* n = new Node();
    n->data = val;
    n->prev = nullptr;
    n->next = nullptr;
    return n;
}

void addOddIndexSumAtEnd(Node* head) {
    int sum = 0, idx = 0;
    Node* cur = head;
    Node* tail = nullptr;

    while (cur) {
        if (idx % 2 == 1) sum += cur->data;
        tail = cur;
        cur = cur->next;
        idx++;
    }

    Node* n = newNode(sum);
    tail->next = n;
    n->prev = tail;
}
```

```
void print(Node* head) {
    for (Node* p = head; p; p = p->next)
        cout << p->data << " ";
}

int main() {
    Node* head = newNode(10);
    head->next = newNode(20);
    head->next->prev = head;
    head->next->next = newNode(30);
    head->next->next->prev = head->next;
    head->next->next->next = newNode(40);
    head->next->next->next->prev = head->next->next;
    head->next->next->next->next = newNode(50);
    head->next->next->next->next->prev = head->next->next->next;

    addOddIndexSumAtEnd(head);
    print(head);
}
```

Input & Output :

```
10 20 30 40 50 60
```

Problem - B:

Code :

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

struct Node {
    int data;
    Node* prev;
    Node* next;
```

```
};

Node* newNode(int val) {
    Node* n = new Node();
    n->data = val;
    n->prev = nullptr;
    n->next = nullptr;
    return n;
}

Node* alternateMerge(Node* h1, Node* h2) {
    Node* head = h1;
    Node* p1 = h1;
    Node* p2 = h2;

    while (p1 && p2) {
        Node* n1 = p1->next;
        Node* n2 = p2->next;

        p1->next = p2;
        p2->prev = p1;

        if (!n1) break;

        p2->next = n1;
        n1->prev = p2;

        p1 = n1;
        p2 = n2;
    }
    return head;
}

void print(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
```

```
    }
}

int main() {
    Node* h1 = newNode(1);
    h1->next = newNode(3); h1->next->prev = h1;
    h1->next->next = newNode(5); h1->next->next->prev = h1->next;
    h1->next->next->next = newNode(7); h1->next->next->next->prev =
h1->next->next;

    Node* h2 = newNode(2);
    h2->next = newNode(4); h2->next->prev = h2;
    h2->next->next = newNode(6); h2->next->next->prev = h2->next;

    Node* merged = alternateMerge(h1, h2);

    cout << "Merged List:\n";
    print(merged);
}
```

Input & Output :

```
Merged List:
1 2 3 4 5 6 7
```

Problem - C:

Code : #include <iostream>
using namespace std;

```
struct Node {
    int data;
    Node* prev;
    Node* next;
};

Node* newNode(int val) {
    Node* n = new Node();
    n->data = val;
    n->prev = nullptr;
    n->next = nullptr;
    return n;
}

void append(Node*& head, Node*& tail, Node* node) {
    if (!head) {
        head = tail = node;
    } else {
        tail->next = node;
        node->prev = tail;
        tail = node;
    }
}

Node* rearrange(Node* head, int low, int high) {
    Node *lessH = nullptr, *lessT = nullptr;
    Node *midH = nullptr, *midT = nullptr;
    Node *moreH = nullptr, *moreT = nullptr;

    while (head) {
        Node* next = head->next;
        head->prev = head->next = nullptr;

        if (head->data < low)
            append(lessH, lessT, head);
        else if (head->data <= high)
```

```
            append(midH, midT, head);
        else
            append(moreH, moreT, head);

        head = next;
    }

    Node* newHead = nullptr;

    if (lessT) {
        newHead = lessH;
        lessT->next = midH ? midH : moreH;
        if (midH) midH->prev = lessT;
        else if (moreH) moreH->prev = lessT;
    } else if (midH) {
        newHead = midH;
    } else {
        newHead = moreH;
    }

    if (midT && moreH) {
        midT->next = moreH;
        moreH->prev = midT;
    }

    return newHead;
}

void print(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
}

int main() {
    Node* head = newNode(1);
```

```
    head->next = newNode(4); head->next->prev = head;
    head->next->next = newNode(3); head->next->next->prev = head->next;
    head->next->next->next = newNode(2); head->next->next->next->prev =
head->next->next;
    head->next->next->next->next = newNode(5);
head->next->next->next->next->prev = head->next->next->next;
    head->next->next->next->next->next = newNode(8);
head->next->next->next->next->next->prev = head->next->next->next->next;
    head->next->next->next->next->next->next = newNode(6);
head->next->next->next->next->next->next->prev =
head->next->next->next->next->next;

    head = rearrange(head, 3, 5);

    cout << "Rearranged List:\n";
    print(head);
}
```

Input & Output :

```
Rearranged List:
1 2 4 3 5 8 6
```