1) Output

```
10 20 30 40
10 20 30 35 40
20 30 40
```

2) List Append:

```
void ListAppend(int elem) {
    // Appends items to the list by setting the tail to point to the added node
    // and the new tail's "next" to nullptr
    // If list is empty it also sets the head to point to the new node
    node* newNode = new node(elem);

    if (head == nullptr) {
        head = newNode;
        tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
}
```

3) List Prepend:

```
void ListPrepend(int elem) {
    // Prepends items to the list by first setting the new node's "next" to the head
    // and only then changing the head pointer to the new node (this order is very important)
    // likewise if the list is empty then it just sets both head and tail to the new node
    node* newNode = new node(elem);
    if (head == nullptr) {
        head = newNode;
        tail = newNode;
    } else {
        newNode->next = head;
        head = newNode;
    }
}
```

4) Search:

```
node* GetNode(int value) {
    // traverses the list until it finds the node with the given value. Returns nullptr if not
    // found
    node* current = head;
    while (current != nullptr) {
        if (current->data == value) {
            return current;
        }
        current = current->next;
    }
    return nullptr;
}
```

5) List Display:

```
void ListDisplay() {
    // prints the data stored in each node as it traverses the list until it reaches the end
    // (the nullptr)
    node* current = head;
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}
```

6) Add After Node:

```
void InsertAfter(node* curNode, int elem) {
    // Inserts a node after another node. If the list is empty, then it inserts it at the
    // beginning If the input is nullptr, it prepends the node to the list,
    // otherwise it adds the new node in front of the old one. If that node is last then it
    // appends it to the list
    node* newNode = new node(elem);

    if (head == nullptr) {
        head = newNode;
        tail = newNode;
        return;
    }

    if (curNode == nullptr) {

        newNode->next = head;
        head = newNode;
        return;
    }

    newNode->next = curNode->next;
    curNode->next = newNode;

    if (curNode == tail) {
        tail = newNode;
    }
}
```

7) Add Before Node:

```
void RemoveAfter(node* curNode) {
    // Similar functionality as above, doesn't do anything if the list is empty,
    // deletes the head if nullptr is given as an argument
    //(making sure to delete the temp) to avoid memory leaks
    // deletes the node after the node given in the argument by appropriately reassinging it's
    // next pointers
    if (head == nullptr) return;

    if (curNode == nullptr) {
        // remove head
        node* temp = head;
        head = head->next;
        if (head == nullptr) {
            tail = nullptr;
        }
        delete temp;
        return;
    }

    node* target = curNode->next;
    if (target != nullptr) {
        curNode->next = target->next;
        if (target == tail) {
            tail = curNode;
        }
        delete target;
    }
}
```