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

struct Node {
    int value;
    Node* next;
};

void printList(Node* list) {
    Node* temp = list;
    while(temp != nullptr) {
        cout << temp->value << " ";
        temp = temp->next;
    }
    cout << endl;
}

// int countFirstValues(Node* list) {
//     if(list == nullptr)
//         return 0;
//     Node* temp = list;
//     int counter = 0;
//     while(temp != nullptr) {
//         if(temp->value == list->value) {
//             counter++;
//         }
//         temp = temp->next;
//     }
//     return counter;
// }
//
// void removeDuplicates(Node* list) {
//     if(list == nullptr )
//         return;
//     Node* temp = list;
//     while(temp->next != nullptr) {
//         if(temp->value == temp->next->value) {
//             Node* save = temp->next;
```

```
39 //             temp->next = save->next; //also = temp
   ->next->next
40 //             delete save;
41 //         }
42 //         else
43 //             temp = temp->next;
44 //     }
45 // }
46 //
47 // Node* extractBetween(Node* list, int startNum, int
    endNum) {
48 //     if(list == nullptr || list->next == nullptr) {
49 //         return nullptr;
50 //     }
51 //     Node* start = nullptr;
52 //     Node* end = nullptr;
53 //     Node* temp = list;
54 //     while(temp->next != nullptr) {
55 //         if(start == nullptr && temp->value ==
   startNum) {
56 //             start = temp;
57 //         }
58 //         else if(start != nullptr && temp->next->
   value == endNum) {
59 //             end = temp;
60 //             Node* save = start->next;
61 //             start->next = end->next;
62 //             end->next = nullptr;
63 //             return save;
64 //         }
65 //         temp = temp->next;
66 //     }
67 //     return nullptr;
68 // }
69 //
70 // bool mostlyEven(Node *list) {
71 //     Node* temp = list;
72 //     int even = 0;
```

```
 73 //     int odd = 0;
 74 //     while(temp != nullptr) {
 75 //         if(temp->value % 2 == 0)
 76 //             even++;
 77 //         else
 78 //             odd++;
 79 //         temp = temp->next;
 80 //     }
 81 //     return even>odd;
 82 // }
 83 // void insertAtBeginning(Node* &list, int num
    ) { // Pass by reference if you want to change it
 84 //     Node* temp = new Node;
 85 //     temp->value = num;
 86 //     temp->next = list;
 87 //     list = temp;
 88 // }
 89 // void moveToFront(Node* &list, int goodNumber) {
 90 //     Node* temp = list;
 91 //     while(temp->next != nullptr) {
 92 //         if(temp->next->value == goodNumber) {
 93 //             insertAtBeginning(list, goodNumber);
 94 //
 95 //             Node* save = temp->next;
 96 //             temp->next = temp->next->next;
 97 //             delete save;
 98 //         }
 99 //         else
100 //             temp = temp->next;
101 //     }
102 // }
103 // void insertAtEnd(Node* &list, int num) {
104 //     if(list == nullptr) {
105 //         list = new Node();
106 //         list->value = num;
107 //         return;
108 //     }
109 //     Node* temp = list;
```

```
110 //      while(temp->next != nullptr) {
111 //          temp = temp->next;
112 //      }
113 //      Node* temp2 = new Node;
114 //      temp2->value = num;
115 //      temp2->next = nullptr;
116 //
117 //      temp->next = temp2;
118 // }
119 //
120 // Node* duplicate(Node* list) {
121 //      if(list == nullptr) {
122 //          return nullptr;
123 //      }
124 //      Node* head = new Node();
125 //      head->next = nullptr;
126 //      head->value = list->value;
127 //      Node* temp = list->next;
128 //      while(temp != nullptr) {
129 //          insertAtEnd(head, temp->value);
130 //          temp = temp->next;
131 //      }
132 //      return head;
133 //
134 // }
135
136 int findBiggest(Node* list) {
137     Node* temp = list;
138     int biggest = -1;
139     if(list == nullptr)
140         return biggest;
141     while(temp != nullptr) {
142         if(temp->value > biggest)
143             biggest = temp->value;
144         temp = temp->next;
145     }
146     return biggest;
147 }
```

```
148
149 void removeOddPositions(Node* list) {
150     if(list == nullptr)
151         return;
152     Node* temp = list;
153     while(temp->next != nullptr) {
154         Node* save = temp->next;
155         temp->next = save->next;
156         if(save->next != nullptr)
157             temp = temp->next;
158         delete save;
159     }
160
161 }
162
163 void modifyList(Node* &list) {
164     if(list == nullptr || list->next == nullptr) {
165         return;
166     }
167
168     Node* temp = list;
169     int value = list->value;
170     int repeat = list->next->value;
171
172     Node* save = temp;
173     list = temp->next;
174     delete save;
175     Node* save2 = list;
176     list = list->next;
177     delete save2;
178
179     for(int i = 0; i < repeat; i++) { //insert at
    beginning
180         Node* insert = new Node;
181         insert->value = value;
182         insert->next = list;
183         list = insert;
184     }
```

```
185 }
186
187 int main() {
188     cout << "Hello, Unit 1 test" << endl;
189     /*tips:
190      * If unsure, make another pointer.
191      * ALWAYS check for nullptr lists
192     */
193
194
195     //read in numbers and make a list
196     Node* head = nullptr;
197     int num;
198     cin >> num;
199     while(num > -1) {
200         Node* temp = new Node();
201         temp->value = num;
202         temp->next = head;
203         head = temp;
204         cin >> num;
205     }
206
207     printList(head);
208
209     //Problem #0
210     int biggestNumber = findBiggest(head);
211     if(biggestNumber == -1)
212         cout << "The list is empty." << endl;
213     else
214         cout << "Biggest number in list is " <<
    biggestNumber << endl;
215
216     //Problem #1
217     cout << "Original List: ";
218     printList(head);
219     removeOddPositions(head);
220     cout << "Removed Odd Positions: ";
221     printList(head);
```

```
222
223     //Problem #2
224     cout << "Value Node: " << head->value << endl;
225     cout << "Repeat Node: " << head->next->value <<
    endl;
226     modifyList(head);
227     printList(head);
228
229 /* 2023 Practice
230     //Problem #0
231     cout << "#0 = " << countFirstValues(head) <<
    endl;
232
233     //Problem #1
234     removeDuplicates(head);
235     cout << "#1 = ";
236     printList(head);
237     cout << endl;
238
239     //Problem #2
240     Node* snip = extractBetween(head, 25, 50);
241     printList(head);
242     printList(snip);
243 */
244 /* 2022 Practice
245     //Problem #0
246     if(mostlyEven(head))
247         cout << "The list is mostly even." << endl;
248     else
249         cout << "The list is not mostly even." <<
    endl;
250
251     //Problem #1
252     moveToFront(head, 9);
253     printList(head);
254
255     //Problem #2
256     printList(duplicate(head));
```

```
257 */
258     return 0;
259 }
260
```