CSc 116 A02 - Fall 2018

Name:

Student Number:

In the space provided below, write a **single** definition for a syntactically correct C++ function `removeIndex` (**including the function signature**) that given a list of **any type** L and an integer index n, will remove the node at n from L and return the value stored that node. You may assume that the list cannot be empty and that the index is always valid. Your solution must work on all inputs, not just the one provided below.

```
#include <iostream>
#include <memory>

template <typename T>
class List {
public:
    List() : head{nullptr}
    {  }

    void push_front(T value) {
        auto node = std::make_shared<Node>(value);
        node->next = head;
        head = node;
    }

    struct Node;
    using NodePtr = std::shared_ptr<Node>;

    struct Node {
        Node(T value) :
            data{value},
            next{nullptr}
        {  }

        T data;
        NodePtr next;
    };

    NodePtr head;
};

// Your function here.

template <typename T>
void printList(List<T> const& list) {
    auto it = list.head;
    while (it != nullptr) {
        std::cout << it->data << " ";
        it = it->next;
    }
    std::cout << std::endl;
}

int main() {
    List<int> list;
    list.push_front(3);
    list.push_front(2);
    list.push_front(1);

    printList(list);
    auto val = removeIndex(1, list);
    printList(list);
    return 0;
}
```

Consider the following possible solution:

```
template <typename T>
T removeIndex(std::size_t i,
    List<T>& list)
{
    auto it = list.head;
    std::size_t index{};

    if (it->next == nullptr)
    {
        list.head = nullptr;
        return it->data;
    }

    while (it->next != nullptr &&
        index < (i - 1))
    {
        it = it->next;
        index++;
    }

    auto pop = it->next;
    it->next = pop->next;
    return it->data;
}
```

When your code is correct, the output will be:

```
1 2 3
1 3
```