

# Linked List Solutions

2020-14-02

Defining the Node class is straightforward.

```
class Node {
public:
    Node(double val, Node *next):
        val_(val), next_(next) { }
    double val_;
    Node *next_;
};
```

The explicitly defined constructor simplifies the code in the LinkedList class, which is given below.

```
class LinkedList {
public:
    LinkedList():
        head_(NULL), size_(0) { }

    double head() const { return head_>val_; }

    int size() const { return size_; }

    void insert(double val) {
        Node *new_node = new Node(val, head_);
        head_ = new_node;
        size_++;
    }

    void remove() {
        Node* temp = head_;
        head_ = head_>next_;
        delete temp;
        size_--;
    }

    void print() const {
        Node *ptr = head_;
        while(ptr != NULL) {
            Rcpp::Rcout << ptr->val_ << " ";
            ptr = ptr->next_;
        }
        Rcpp::Rcout << std::endl;
    }

private:
    Node *head_;
    int size_;
};
```

```
};
```

This linked list object is initialised to an empty list, whereby `head_` is a null pointer. We have provided a simple accessor `LinkedList::head()` to the data of the node pointed to by `head_`. **This method is unsafe:** if called when `head_` is null, it will lead to a segfault. We do not implement validation here, and instead do this at the R level. We assume the class is used by a competent developer (you), and not by the end user. Similarly, `LinkedList::Remove()` is unsafe for the same reasons (check the code).

Exposing this class to R using Rcpp modules is straightforward. Consider the code below.

```
Rcpp_MODULE(LinkedList_module) {  
  Rcpp::class_<LinkedList>("LinkedList")  
  
  .constructor()  
  .method("insert", &LinkedList::insert, "Adds a double value to the list.")  
  .method("remove", &LinkedList::remove, "Deletes current head of the list.")  
  .method("head", &LinkedList::head, "Returns value of the head node of the list.")  
  .method("print", &LinkedList::print, "Simple printing of all values in the list")  
  .method("size", &LinkedList::size, "Returns total number of elements in the list.")  
  ;  
}
```

This defines an R class, which will be called “LinkedList” in R. It also defines an S4 class called “Rcpp\_LinkedList” in R. This is useful for S4 dispatch - a topic we will come to later. All public methods of the `LinkedList` class are exposed, in addition to the constructor.

Assume all of the above code is in a module called “linkedlist.cpp”. We can make the class available in R using the following code in the R script.

```
Rcpp::sourceCpp("scripts/linkedlist.cpp")
```

```
## Warning: package 'Rcpp' was built under R version 3.5.2
```