Universitetet
i Sørøst-Norge

Implementasjon av dobbeltlenket sirkulær liste i C++

Christian McGloin

Sep 2024

USN

Prog 3

1 Introduction

This document presents the implementation of a doubly linked circular list in C++. The requirements for this task are to have a function that adds an element to the list; the list shall be sorted based on ID. Another function will remove a node without breaking the circular list. Then, a final function will print out all of the elements in the list and ensure that we are not printing out the elements in a loop. Some more requirements: there shall not be more than one element with the same ID; the remove function shall delete all nodes with the same name. If there are more than one node with the same name, then all shall be removed. Finally, one more function that prints out all the names, sorted.

2 Implementation

I have chosen for the class **DobbeltlenketSirkulærListe()** to have four public methods and one private method, and to implement the node as a struct instead of as a separate class with a friend class, as shown in the course book [1].

leggtil(): This function ensures that we keep IDs unique and takes **Navn** as a string input. It checks if the list is empty, and if not, adds it to the correct place in the list and rearranges the previous node.

skrivUtNavnAlfabetisk(): We start by breaking the circularity of the list to make it easier for sorting, building a new sorted list, and going through each node in the original list. We compare by using the function **insertSorted()** to place the node in its sorted place. When the new sorted list is finished, we need to restore it to a doubly linked circular list.

SkrivUt(): Chose to use this function for printing the current list to the screen, for both the original and sorted list, by using a do-while loop.

slett(): I went with function overloading to handle both **int** and **string** input. The **string** function was a bit more challenging because it needs to be able to loop through and delete multiple entries. I chose to handle this by keeping track of the starting point, so when a match at **current** is deleted and pointers are updated, we go through the list again until there are no more matches.

3 Results

main(): Contains the results we were looking for and confirms the program can handle each requirement for our program, showing the list sorted by **ID**, being able to delete by **ID** number, handling deleting a node that started as head and contains multiple entries, and finally **skrivUtNavnAlfabetisk** to sort the list alphabetically.

4 Conclusion

The most challenging part was implementing the sorting algorithm and being able to delete multiple entries in **slett(std::string)**. Initially, I tried to use the same logic as for **slett(int)**, but I was unable to find a solution for multiple entries that were next to each other. If I were to do this again, I would try to use a more efficient sorting algorithm, like Merge Sort.

Listing 1: Code

```
#include <iostream>
#include <string>
class DobbeltlenketSirkularListe {
public:
    void leggtil(std::string Navn);
    void skrivUt();
    void slett(int id);
    void slett(std::string Navn);
    void skrivUtNavnAlfabetisk();
private:
```

```
    int next_id = 1;
    struct Node {
        int id;
        std::string Navn;
        std::shared_ptr<Node> next;
        std::shared_ptr<Node> prev;
        Node(int id, std::string Navn) : id(id), Navn(Navn) {}
    };
    std::shared_ptr<Node> head = nullptr;
    void insertSorted(std::shared_ptr<Node>& sorted, std::shared_ptr<Node> newNode
        );
};
void DobbeltlenketSirkularListe::leggtil(std::string Navn){
    int id = next_id++;
    std::shared_ptr<Node> newNode = std::make_shared<Node>(id, Navn);
    if (head == nullptr) {
        head = newNode;
        head->next = head;
        head->prev = head;
    }
    else {
        std::shared_ptr<Node> tail = head->prev;
        tail->next = newNode;
        newNode->prev = tail;
        newNode->next = head;
        head->prev = newNode;
    }
}
void DobbeltlenketSirkularListe::insertSorted(std::shared_ptr<Node>& sorted, std::
    shared_ptr<Node> newNode) {
    if (!sorted) {
        sorted = newNode;
        return;
    }
    if (newNode->Navn < sorted->Navn) {
        newNode->next = sorted;
        sorted->prev = newNode;
        sorted = newNode;
        return;
    }
    std::shared_ptr<Node> current = sorted;
    while (current->next != nullptr && current->next->Navn < newNode->Navn) {
        current = current->next;
    }
    newNode->next = current->next;
    if (current->next != nullptr) {
        current->next->prev = newNode;
    }
    current->next = newNode;
    newNode->prev = current;
}
void DobbeltlenketSirkularListe::skrivUtNavnAlfabetisk() {
    if (!head || head->next == head) {return;}
    std::shared_ptr<Node> tail = head->prev;
    tail->next = nullptr;
    head->prev = nullptr;
    std::shared_ptr<Node> sorted = nullptr;
    std::shared_ptr<Node> current = head;
    while (current != nullptr) {
```

```
        std::shared_ptr<Node> nextNode = current->next;
        current->next = nullptr;
        current->prev = nullptr;
        insertSorted(sorted, current);
        current = nextNode;
    }
    if (sorted != nullptr) {
        std::shared_ptr<Node> sortedTail = sorted;
        while (sortedTail->next != nullptr) {
            sortedTail = sortedTail->next;
        }
        sortedTail->next = sorted;
        sorted->prev = sortedTail;
        head = sorted;
    }
}
void DobbeltlenketSirkularListe::skrivUt(){
    if (!head) return;
    std::shared_ptr<Node> current = head;
    do {
        std::cout << "ID: " << current->id << ", Name: " << current->Navn << std::
            endl;
        current = current->next;
    } while (current != head);
}
void DobbeltlenketSirkularListe::slett(int a) {
    if (!head) return;
    std::shared_ptr<Node> current = head;
    do {
        if (current->id == a) {
            std::cout << "ID " << a << " Slettet" << std::endl;
            std::shared_ptr<Node> uPrev = current->prev;
            std::shared_ptr<Node> wNext = current->next;
            uPrev->next = wNext;
            wNext->prev = uPrev;
            if (current == head) {
                if (current->next == current) {
                    head = nullptr;
                }
                else {
                    head = wNext;
                }
            }
            return;
        }
        current = current->next;
    } while (current != head);
}
void DobbeltlenketSirkularListe::slett(std::string a) {
    if (!head) return;
    std::shared_ptr<Node> current = head;
    std::shared_ptr<Node> start = head;
    bool firstIteration = true;
    while (head != nullptr) {
        if (current->Navn == a) {
            std::cout << "Navn " << a << " Slettet" << std::endl;
            std::shared_ptr<Node> uPrev = current->prev;
            std::shared_ptr<Node> wNext = current->next;
            uPrev->next = wNext;
```

```
            wNext->prev = uPrev;
            if (current == head) {
                if (current->next == current) {
                    head = nullptr;
                    return;
                }
                else {
                    head = wNext;
                }
            }
            current = head;
            start = head;
            firstIteration = true;
        }
        else {
            current = current->next;

            if (current == start) {
                if (firstIteration) {
                    firstIteration = false;
                }
                else {
                    break;
                }
            }
        }
    }
}
int main() {
    DobbeltlenketSirkularListe liste;
    liste.leggtil("Julenissen");
    liste.leggtil("Silje");
    liste.leggtil("Jens");
    liste.leggtil("Jens");
    liste.leggtil("Maria");
    liste.leggtil("Christian");
    liste.leggtil("Sigrid");
    liste.leggtil("Jens");
    liste.leggtil("Emily");
    std::cout << "liste usortert, ID lavt til hoyt: " << std::endl;
    liste.skrivUt();
    liste.slett(1);
    liste.skrivUt();
    liste.slett(std::string("Jens"));
    liste.skrivUt();
    std::cout << "Sorterer listen alfabetisk  " << std::endl;
    liste.skrivUtNavnAlfabetisk();
    liste.skrivUt();
    return 0;
}
```

References

[1] Michael T. Goodrich, Roberto Tamassia, and David M. Mount. *Data Structures and Algorithms in C++, Second Edition*. Wiley, 2011, Example 3.28.