Phill-DS-0920

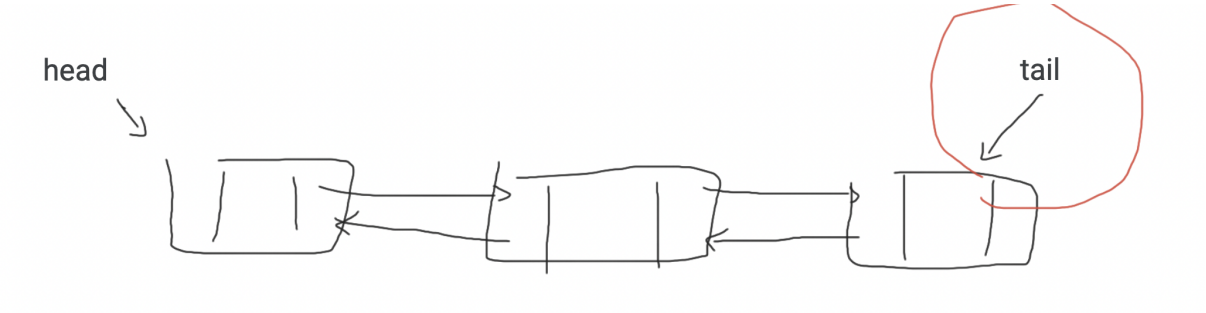
單向 singly linked list

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

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


class LinkedList{
  private:
    Node* head;

  public:
    LinkedList(){
      head=nullptr;
    }

    void add(int data){
      Node* newNode = new Node{data, nullptr};
      if(head==nullptr){ //串列是空的
        head=newNode;
      }
      else{ //有 node inside
        Node* current=head; //產生新的魁儡變數 current
        while(current->next!=nullptr){ //還沒到底
          current = current -> next; //連下一個
        }
        current -> next = newNode;//把自己接到尾巴後面
      }
    }

    void print(){
      Node* current=head;
      while(current != nullptr){
          cout << current->data << "->";
          current = current -> next;
      }
      cout << "nullptr" << endl;
    }

    Node* search(int data){
      Node* current = head;
      while(current !=nullptr){
        if(current->data==data){ // hit data
          return current; //把節點傳回去
        }
```

```
        current = current ->next; //繼續找
      }
      return nullptr; //什麼都沒找到, 傳 null
    }

    void update(int oldData, int newData){
      Node* node = search(oldData); //接住找到的節點
      if(node != nullptr){ // 檢查清楚 防止沒找到的情況 !!
        node-> data = newData; // in lvalue -> 賦值
      }
    } // return bool 當作業

    void remove(int data){
      Node* current = head;
      Node* previous = nullptr;

      while(current != nullptr){
        if(current -> data == data){ //找到了
          if( previous== nullptr){ //代表現在還沒 bookeeping, 是頭節點
              head = current-> next; //直接把頭指到我的下一個 因我要被刪掉
          }
          else{ //不是頭節點的狀況 做連接手術
              previous->next = current -> next; //刪除 接續 把上一個直接接到下一個
          }
          delete current; //處理自己空間的釋放
          return; //返回
        }
        previous = current; //把自己記起來 因為我是下一個人的 previous, bookeeping
        current = current -> next; //放心繼續找下去
      }
    } // bool version -> 當作業

    ~LinkedList(){
      Node* current =head;
      while(current!=nullptr){
          Node* temp = current; //先紀錄我的位址
          current = current -> next; //先把魁儡變數移到下一個人 避免我被release 掉消失
          delete temp; // 釋放自己
      }
    }
};

int main()
{// 設備管理清單
    LinkedList equipIds;
    equipIds.add(101);
    equipIds.add(102);
    equipIds.add(103);

    equipIds.print();//加成功了

    equipIds.update(102, 202);
    equipIds.print();

    equipIds.remove(101); //試特殊的頭
    equipIds.print();
```

```
    return 0;
}
```

doubly linked list




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

struct Node{
  int data;
  Node* next;
  Node* prev; //雙向連結 可向上連
};

class DoublyLinkedList {
  private:
    Node* head;
    Node* tail; //因為可以向前，加 tail 可以從末端開始更快

  public:
    DoublyLinkedList(){
      head = nullptr;
      tail = nullptr;
    }

    void add(int data){
      Node* newNode = new Node{data, nullptr, nullptr};

      if (head==nullptr){
          //沒有 node 存在
          head = newNode;
          tail = newNode;
      }
      else{
        //有 node 存在
        tail->next = newNode; //加到尾
        newNode -> prev = tail; //元素本身 也要把tail 加入雙向鏈結
        tail = newNode; // 把尾指向新元素
      }
    }

    void print(){
```

```
      Node* current = head;
      while(current != nullptr){
        cout << current-> data << "<->";
        current = current -> next;
      }
      cout << "null" << endl;
    }

    Node* search(int data){
      Node* current = head;
      while(current!= nullptr){// traverse
        if(current->data == data){ //找到了
          return current;
        }
        current = current->next;
      }
      return nullptr; //找不到
    }

    void update(int oldData, int newData){
      Node* node = search(oldData);
      if(node != nullptr){ //要檢查 -> 找不到的 case
        node->data = newData; //更新
      }
    }

    void remove(int data){
      Node* current = search(data); //找到資料刪除
      if(current != nullptr){ //避免是找不到的情況
        if(current->next!= nullptr){//後面有連到節點
          //把下一個的 prev 串到前一個
          current->next->prev = current -> prev;
        }
        if(current->prev!= nullptr){//前面真的有東西
          //把上一個的 next 串到下一個
          current->prev->next = current -> next;
        }
        if(current== head){ //head 指到自己 , 我就是 第一個
          head = current -> next; //直接把 head 指向我後面, 我就不在連結上了
        }
        if(current== tail){ //tail 指到自己 , 我就是 最後一個
          tail = current -> prev; //直接把 tail 指向我前面, 我就不在連結上了
        }

        delete current;
      }
    }

};



int main()
{
    DoublyLinkedList list;
```

```
    list.add(9487);
    list.add(9488);
    list.add(9489);

    list.print();

    list.remove(9488);
    list.print();

    return 0;
}
```