Phill DS-0927

```
#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;
      }
    }

};
```

練習 play list

- “不為誰而做的歌”
- “修煉愛情“
- “江南”

```
//修改 class  -> data 從 int 改成 string

int main()
{
    DoublyLinkedList Playlist ;
```

```
    Playlist.add("不為誰而做的歌");
    Playlist.add("修煉愛情");
    Playlist.add("江南");
    Playlist.print();


    Playlist.update("修煉愛情","小酒窩");
    Playlist.print();

    Playlist.remove("江南");
    Playlist.print();


    return 0;
}
```

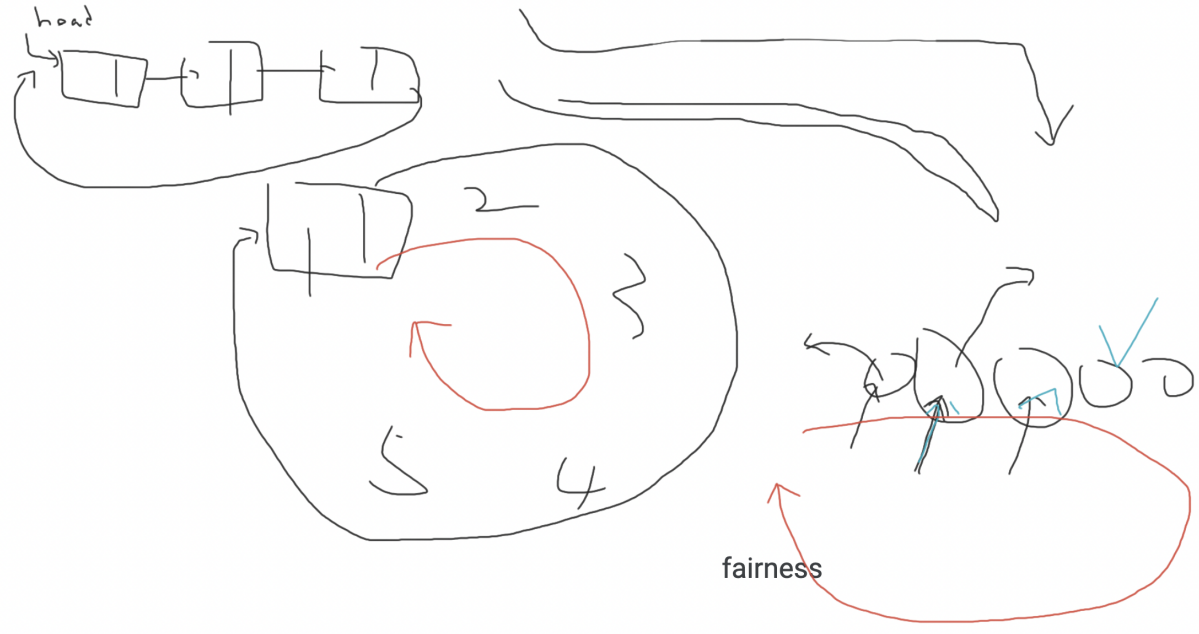
練習 瀏覽器

- class 做修改
- URL → 輸入網址
- URL*3
- back 上一頁
- forward 下一頁
    - 輸入 URL
    - print URL
    - 輸入 back, 呼叫 back()
    - 跳到 上一頁 印出上一頁的 URL
    - 輸入 forward , forward()
    - 跳到 下一頁 印出下一頁的 URL

```

```

環狀 Circular Linked List



- traverse endless 無限制遍歷
- Fairness 程序process分配公平性

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

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

class CircularLinkedList{
  private:
    Node* head;

  public:
    CircularLinkedList(){
      head= nullptr;
    }

    void add(int data){
      Node* newNode= new Node{data, nullptr};
      if(head== nullptr){ // 鍊表為空
        head = newNode; //把自己加入 head
        newNode->next = head; //元素只有一個 把自己指回自己(環狀)
      }
      else{ //已經有 node 在裡面, 加尾巴
        Node* current=head; //設定魁儡變數在開頭
        while(current->next != head){ //環狀特徵, 還不到底
          current = current -> next; //指向下一個
        }
        //已指向最後一個
        current -> next = newNode; //把自己加在尾巴
```

```
         newNode -> next = head; //環狀指回
       }

    }
};



int main()
{
    CircularLinkedList list;
    list.add(123);
    list.add(456);

    return 0;
}
```