Phill-DS-1005

browser answer

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

struct Node{
  string 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
    Node* now;

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

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

      if (head==nullptr){
          //沒有 node 存在
          head = newNode;
          tail = newNode;
          //for update current browser URL
          now = 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
        // chromeURL do update
        now = newNode;
      }
    }
```

```
void forward(void){ // chrome do forward
  if(now==nullptr) return; // 空的
  if(now==tail) {//到底了 無法再forward
    now=tail;
    return;
  }
  now = now -> next;

}

void backward(void){
  if(now==nullptr) return; //空的
  if(now==head) { //到頭了 無法 backward
    now=head;
    return;
  }
  now = now -> prev;
}

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

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


/*
```

```
    void remove(string 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 chromeHistory;

  chromeHistory.add("google.com");
  chromeHistory.add("openai.com");
  chromeHistory.add("tesla.com");

  chromeHistory.print();

  //上一頁
  chromeHistory.backward();
  chromeHistory.print();
  //上一頁
  chromeHistory.backward();
  chromeHistory.print();
  //不能再上一頁了 (boundary test)
  chromeHistory.backward();
  chromeHistory.print();

  //下一頁
  chromeHistory.forward();
  chromeHistory.print();

  //下一頁
  chromeHistory.forward();
  chromeHistory.print();
```

```
  //不能再下一頁了 (boundary test)
  chromeHistory.forward();
  chromeHistory.print();


}
```

Stack

- LIFO 後進先出
- 實作
  - array → 簡單; 不容易出錯(pointer)
  - linked list → 資料本身很複雜

陣列版

```
#include <iostream>
#define MAX_SIZE 20 //陣列容量
using namespace std;

class StackArray{
  private:
    int arr[MAX_SIZE]; //陣列
    int top; // 指向最上面的元素, 以便 pop 出來, 內容應為 index

  public:
    StackArray():top(-1){}

    void push(int data){ //把 data push 進 StackArray
      if(top > MAX_SIZE-1){
        cout << "Overflow" << endl;
        return;
      }
      arr[++top] = data; //先遞增top(從 index=0 開始存) 再存入資料
    }

    int pop(void){ //照順序彈出, 不需要給參數
      if(top <0){ //真的是空了
        cout << "underflow"<< endl;
        return -9999; //return 一個錯誤值
      }
```

```
      return arr[top--]; //-- 克服掉return 後無法處理的問題
    }

    void print(){
      cout << "[";
      for(int i=0; i<=top; i++){
        cout << arr[i] << " " ;
      }
      cout << endl;

    }
    bool isEmpty(void){
      return (top==-1)?true:false;
    }
};


int main()
{
    StackArray abc;
    abc.push(10);
    abc.push(20);
    abc.push(30);

    abc.print();

    cout << "pop->" << abc.pop()<< endl;
    cout << "pop->" << abc.pop()<< endl;

    abc.print();
    return 0;
}
```