資料結構與C++進階班
鏈結串列

講師：黃銀鵬

E-mail: yinpenghuang@gmail.com

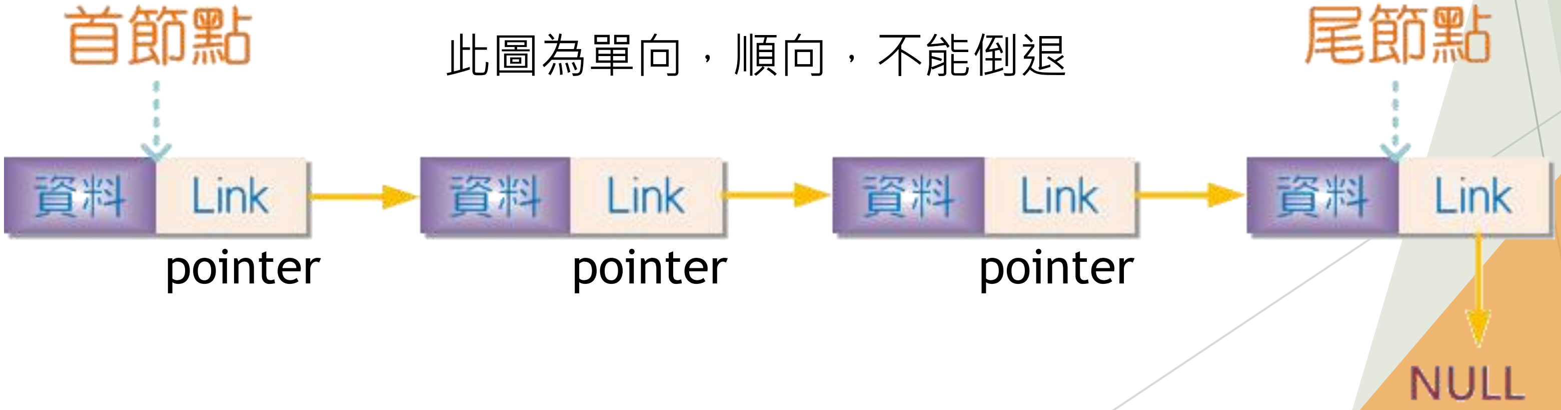
節點靠指標連接，能不能自己實作出鏈結串列是可以評斷你是否懂指標，細細碎碎到處配

鏈結串列

若要解構怎麼辦?
要怎麼刪除節點:

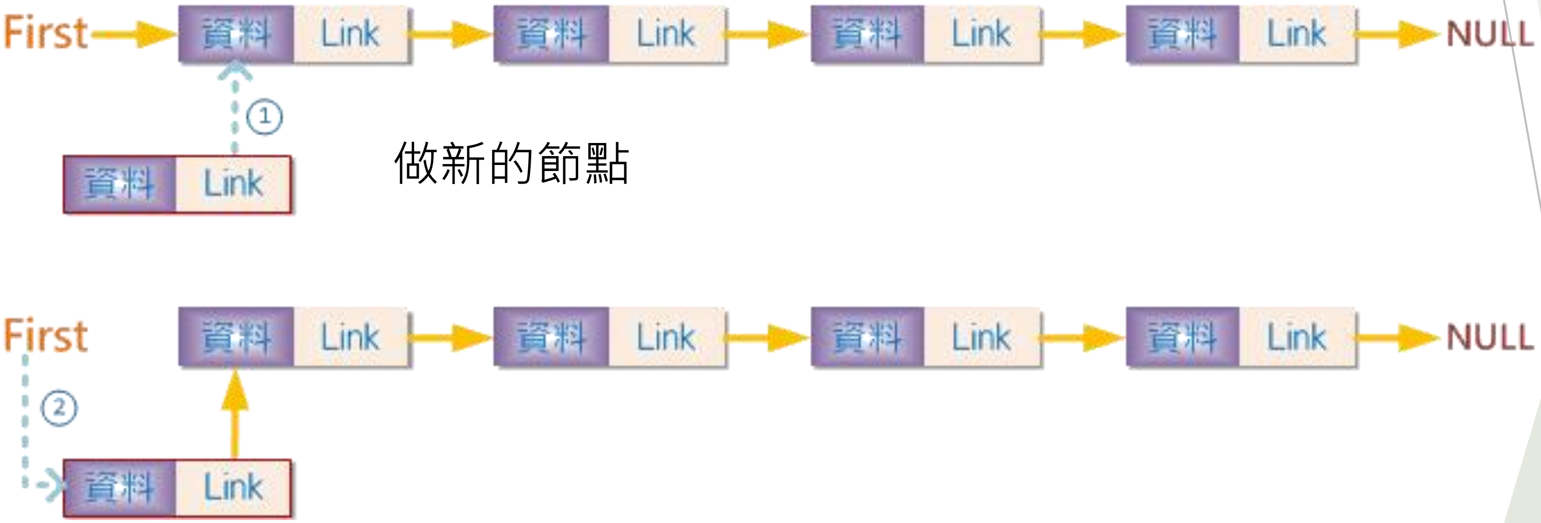
堆疊、佇列(陣列式)	鏈結串列
支援隨機存取	僅循序存取(要從第0個開始找)
頭尾新增刪除元素快	頭尾新增刪除元素快
中間新增刪除元素慢	中間新增刪除元素快
使用較少記憶體	使用較多記憶體
動態增長空間較耗時	動態增長空間較省時

此圖為單向，順向，不能倒退



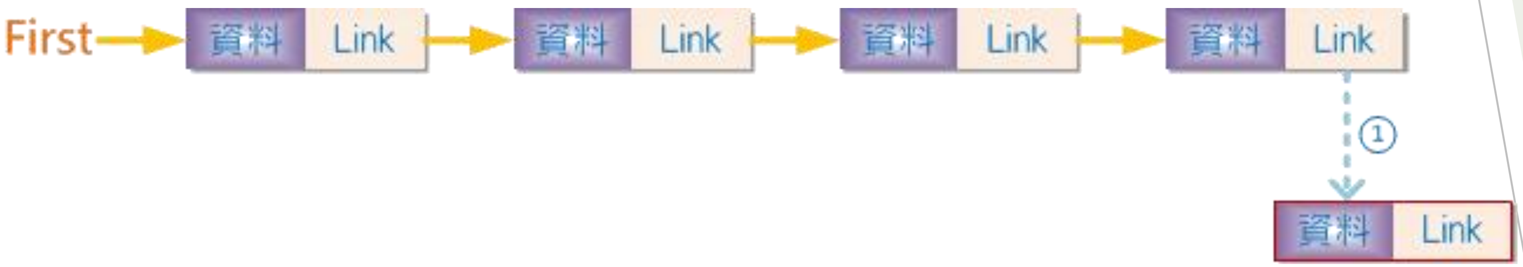
pointer pointer pointer

插入資料-前端插入

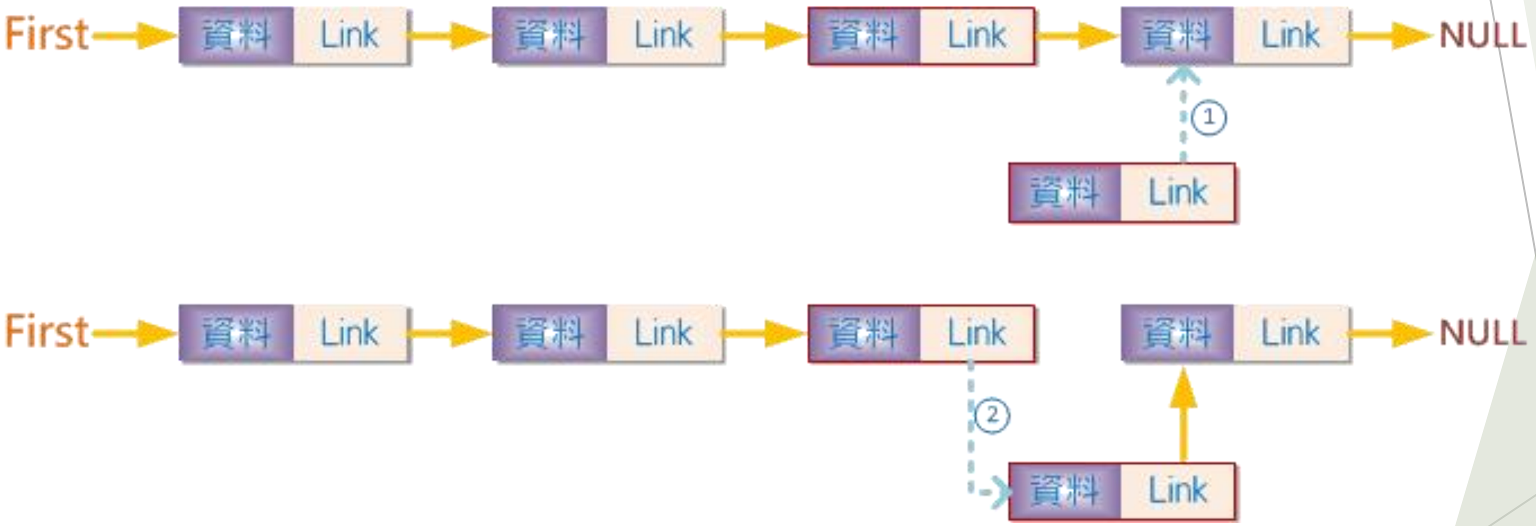


做新的節點

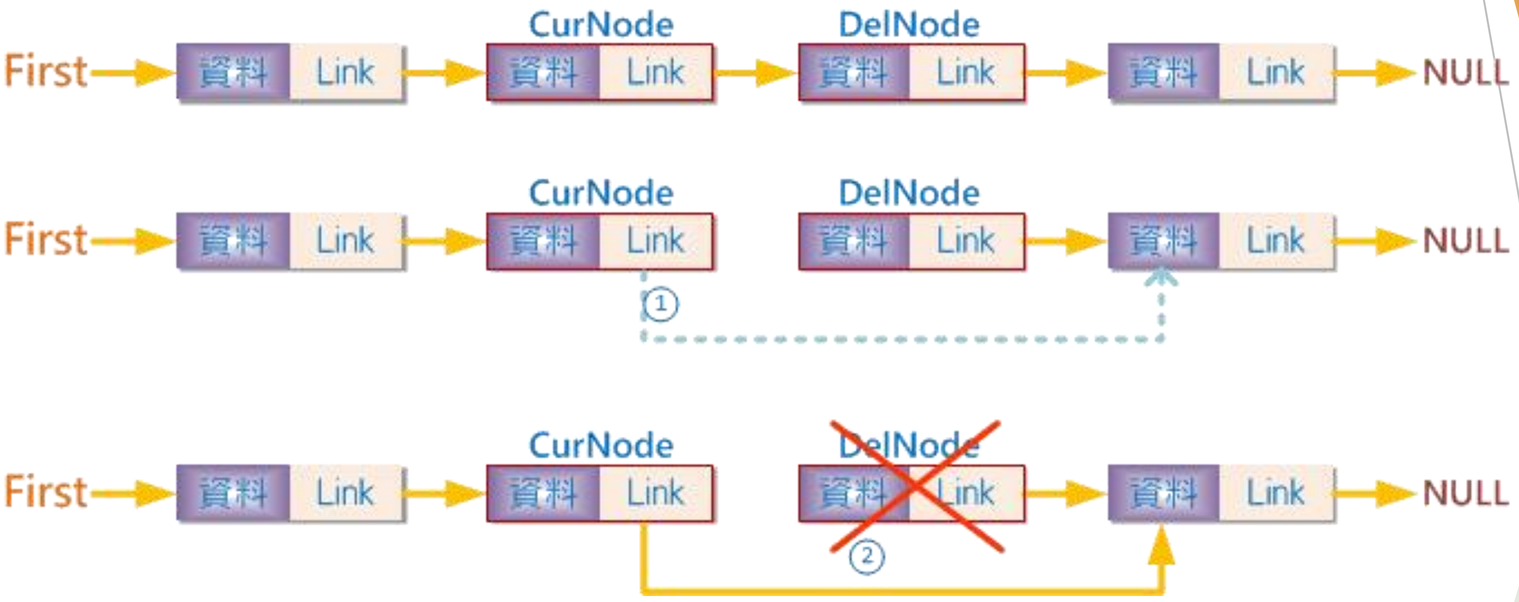
插入資料-後端插入



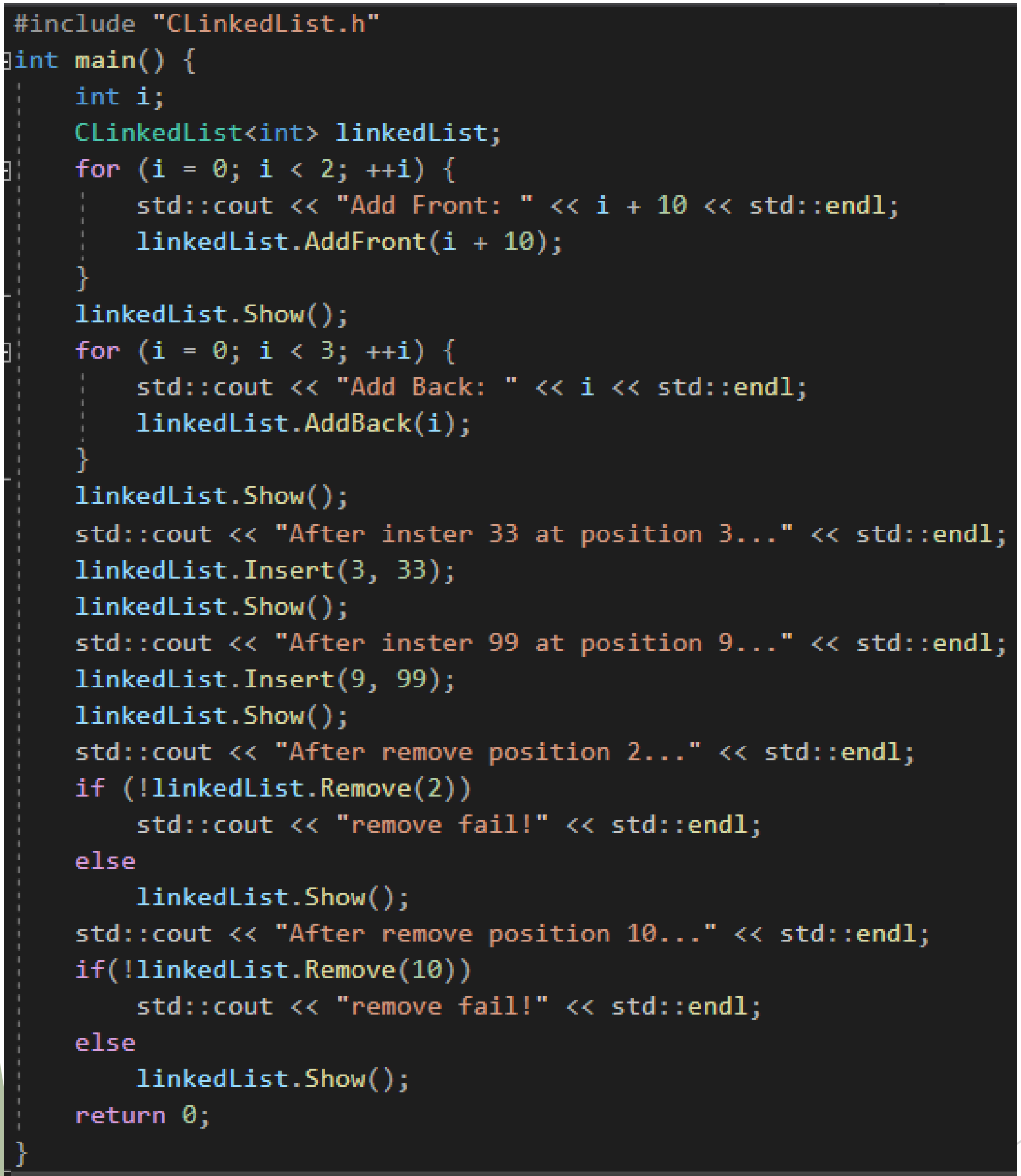
插入資料-中間插入


First
資料
Link
資料
Link
資料
Link
資料
Link
NULL
①
資料
Link
First
資料
Link
資料
Link
資料
Link
資料
Link
NULL
②
資料
Link

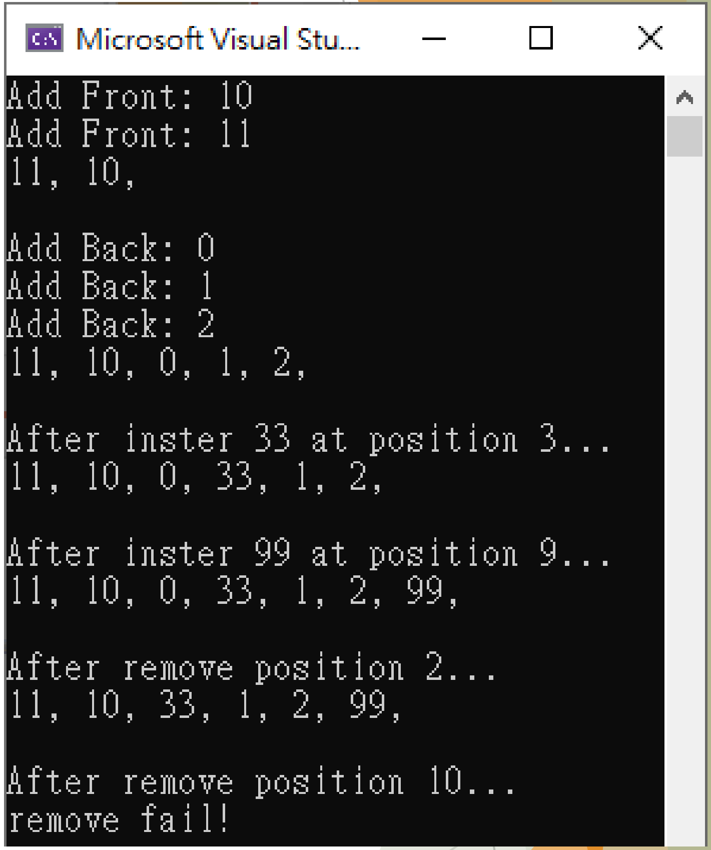
移除資料


CurNode
DelNode
First
資料
Link
資料
Link
資料
Link
資料
Link
NULL
CurNode
DelNode
First
資料
Link
資料
Link
資料
Link
資料
Link
NULL
①
CurNode
DelNode
First
資料
Link
資料
Link
資料
Link
資料
Link
NULL
②

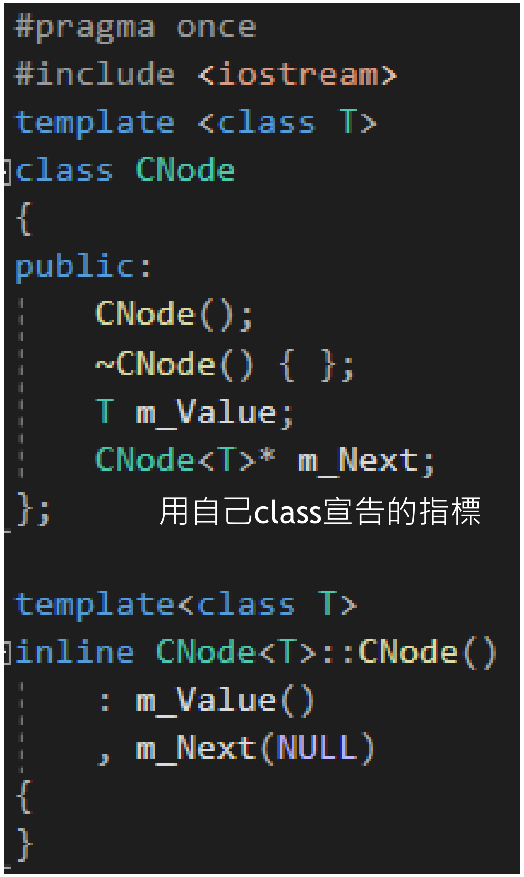
單向鏈結串列



```cpp
#include "CLinkedList.h"
int main() {
    int i;
    CLinkedList<int> linkedList;
    for (i = 0; i < 2; ++i) {
        std::cout << "Add Front: " << i + 10 << std::endl;
        linkedList.AddFront(i + 10);
    }
    linkedList.Show();
    for (i = 0; i < 3; ++i) {
        std::cout << "Add Back: " << i << std::endl;
        linkedList.AddBack(i);
    }
    linkedList.Show();
    std::cout << "After inster 33 at position 3..." << std::endl;
    linkedList.Insert(3, 33);
    linkedList.Show();
    std::cout << "After inster 99 at position 9..." << std::endl;
    linkedList.Insert(9, 99);
    linkedList.Show();
    std::cout << "After remove position 2..." << std::endl;
    if (!linkedList.Remove(2))
        std::cout << "remove fail!" << std::endl;
    else
        linkedList.Show();
    std::cout << "After remove position 10..." << std::endl;
    if(!linkedList.Remove(10))
        std::cout << "remove fail!" << std::endl;
    else
        linkedList.Show();
    return 0;
}
```

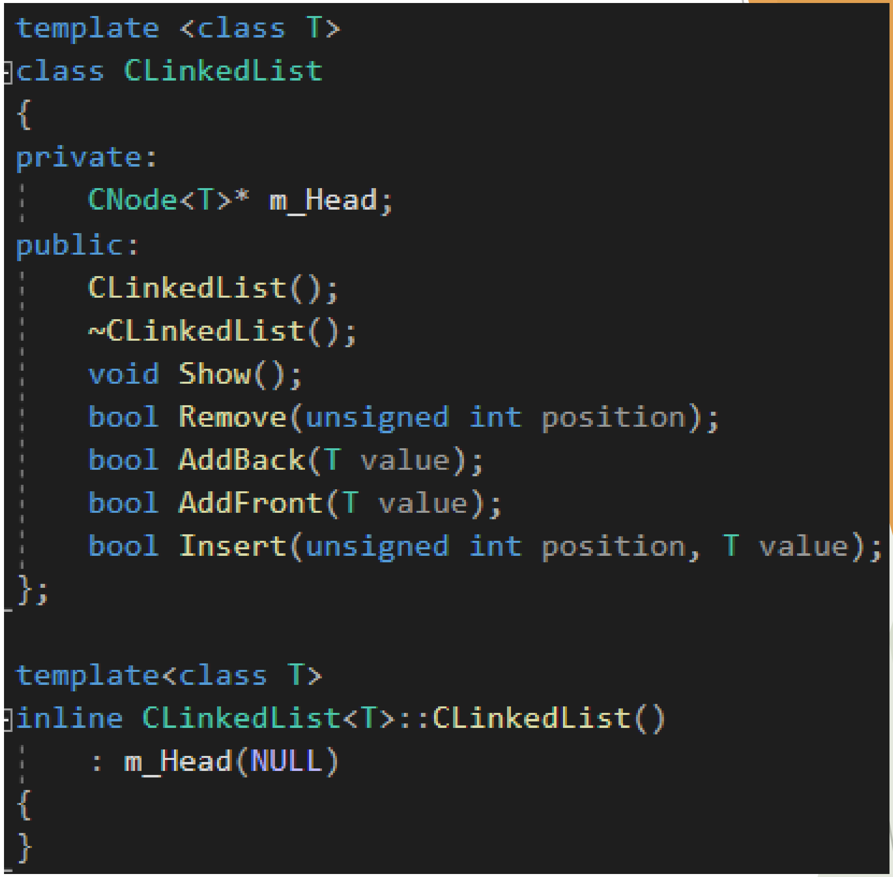


```
Add Front: 10
Add Front: 11
11, 10,

Add Back: 0
Add Back: 1
Add Back: 2
11, 10, 0, 1, 2,

After inster 33 at position 3...
11, 10, 0, 33, 1, 2,

After inster 99 at position 9...
11, 10, 0, 33, 1, 2, 99,

After remove position 2...
11, 10, 33, 1, 2, 99,

After remove position 10...
remove fail!
```



```cpp
#pragma once
#include <iostream>
template <class T>
class CNode
{
public:
    CNode();
    ~CNode() { };
    T m_Value;
    CNode<T>* m_Next;
};

template<class T>
inline CNode<T>::CNode()
    : m_Value()
    , m_Next(NULL)
{
}
```

用自己class宣告的指標



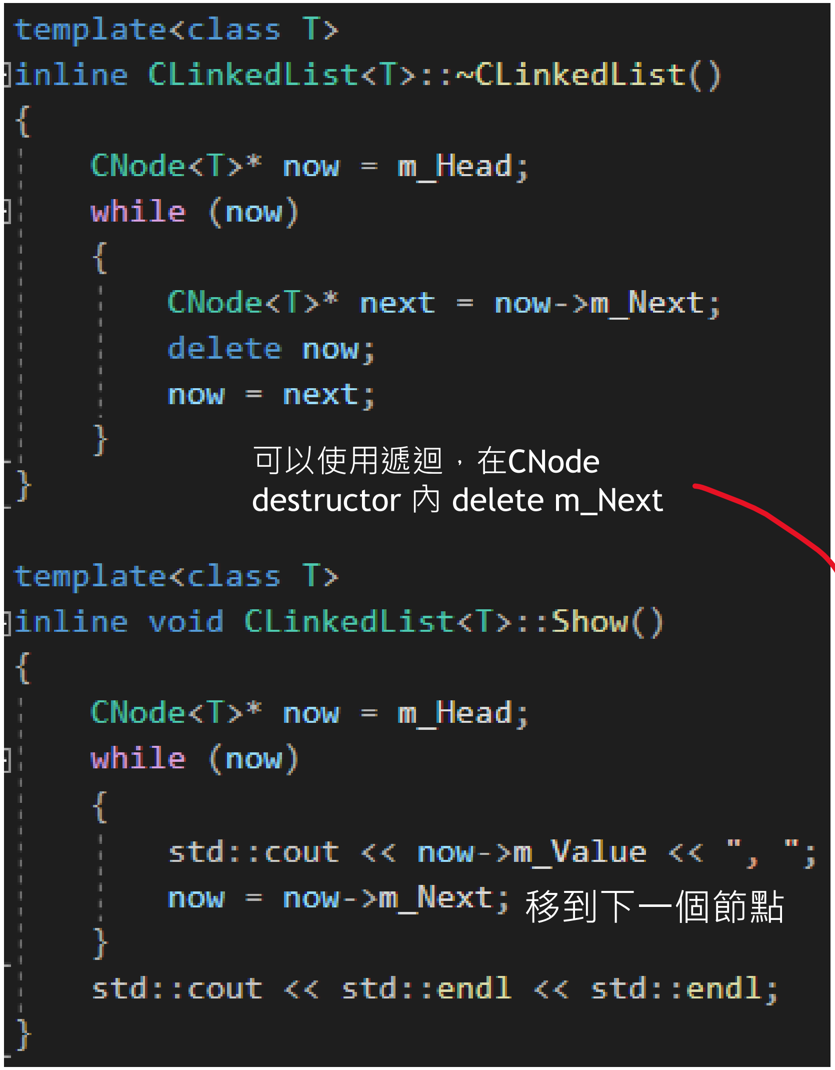
```cpp
template <class T>
class CLinkedList
{
private:
    CNode<T>* m_Head;
public:
    CLinkedList();
    ~CLinkedList();
    void Show();
    bool Remove(unsigned int position);
    bool AddBack(T value);
    bool AddFront(T value);
    bool Insert(unsigned int position, T value);
};

template<class T>
inline CLinkedList<T>::CLinkedList()
    : m_Head(NULL)
{
}
```

只有變數的話，就用struct就好啦，為何還要用class呢?

因為struct 沒辦法支援template

template<此處 T>

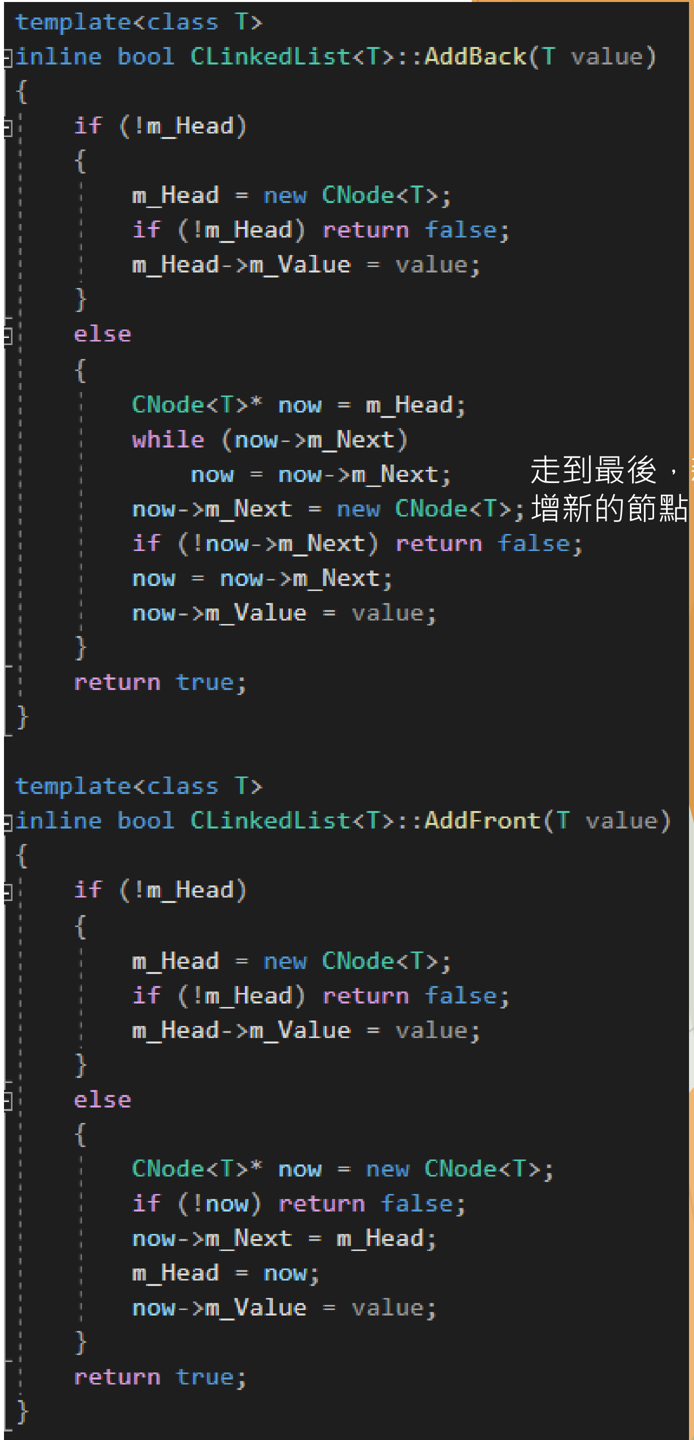
此處雖然可以叫struct，但跟上面是兩回事



```cpp
template<class T>
inline CLinkedList<T>::~CLinkedList()
{
    CNode<T>* now = m_Head;
    while (now)
    {
        CNode<T>* next = now->m_Next;
        delete now;
        now = next;
    }
}

template<class T>
inline void CLinkedList<T>::Show()
{
    CNode<T>* now = m_Head;
    while (now)
    {
        std::cout << now->m_Value << ", ";
        now = now->m_Next;
    }
    std::cout << std::endl << std::endl;
}
```

可以使用遞迴，在CNode destructor 內 delete m_Next

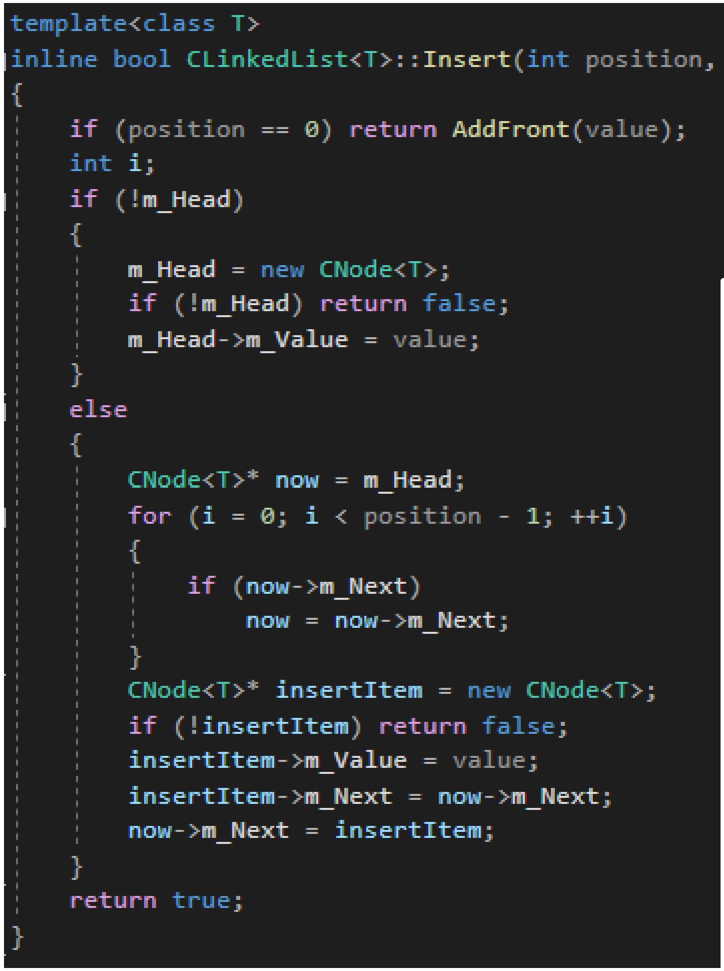
移到下一個節點



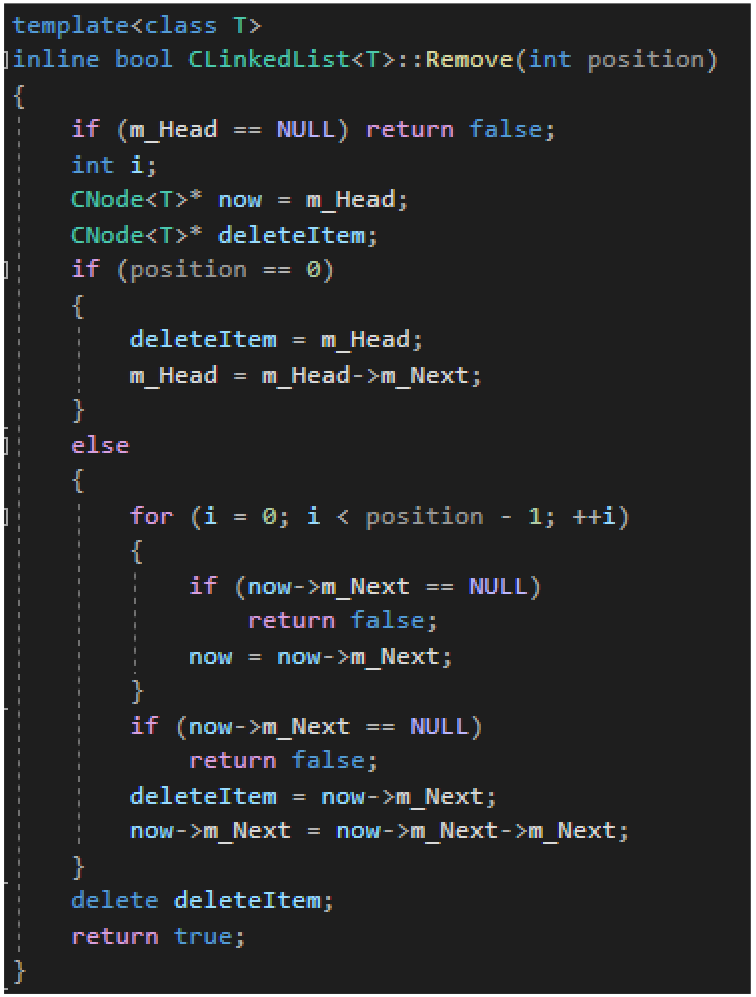
```cpp
template<class T>
inline bool CLinkedList<T>::AddBack(T value)
{
    if (!m_Head)
    {
        m_Head = new CNode<T>;
        if (!m_Head) return false;
        m_Head->m_Value = value;
    }
    else
    {
        CNode<T>* now = m_Head;
        while (now->m_Next)
            now = now->m_Next;
        now->m_Next = new CNode<T>;
        if (!now->m_Next) return false;
        now = now->m_Next;
        now->m_Value = value;
    }
    return true;
}

template<class T>
inline bool CLinkedList<T>::AddFront(T value)
{
    if (!m_Head)
    {
        m_Head = new CNode<T>;
        if (!m_Head) return false;
        m_Head->m_Value = value;
    }
    else
    {
        CNode<T>* now = new CNode<T>;
        if (!now) return false;
        now->m_Next = m_Head;
        m_Head = now;
        now->m_Value = value;
    }
    return true;
}
```

走到最後，新增新的節點

但有缺點，就是執行流程若離開建構子或解構子，一離開出了甚麼差錯，如果回不來呢?就會導致建構不完全，或解構不完全，危險呀!!!



```cpp
template<class T>
inline bool CLinkedList<T>::Insert(int position, T value)
{
    if (position == 0) return AddFront(value);
    int i;
    if (!m_Head)
    {
        m_Head = new CNode<T>;
        if (!m_Head) return false;
        m_Head->m_Value = value;
    }
    else
    {
        CNode<T>* now = m_Head;
        for (i = 0; i < position - 1; ++i)
        {
            if (now->m_Next)
                now = now->m_Next;
        }
        CNode<T>* insertItem = new CNode<T>;
        if (!insertItem) return false;
        insertItem->m_Value = value;
        insertItem->m_Next = now->m_Next;
        now->m_Next = insertItem;
    }
    return true;
}
```



```cpp
template<class T>
inline bool CLinkedList<T>::Remove(int position)
{
    if (m_Head == NULL) return false;
    int i;
    CNode<T>* now = m_Head;
    CNode<T>* deleteItem;
    if (position == 0)
    {
        deleteItem = m_Head;
        m_Head = m_Head->m_Next;
    }
    else
    {
        for (i = 0; i < position - 1; ++i)
        {
            if (now->m_Next == NULL)
                return false;
            now = now->m_Next;
        }
        if (now->m_Next == NULL)
            return false;
        deleteItem = now->m_Next;
        now->m_Next = now->m_Next->m_Next;
    }
    delete deleteItem;
    return true;
}
```

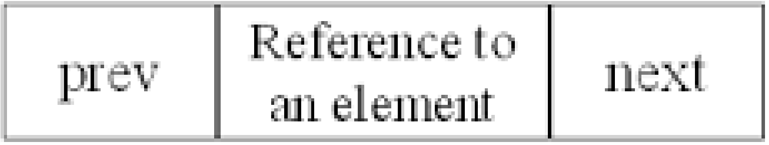
停在要插入的
前一個，或最
後一個

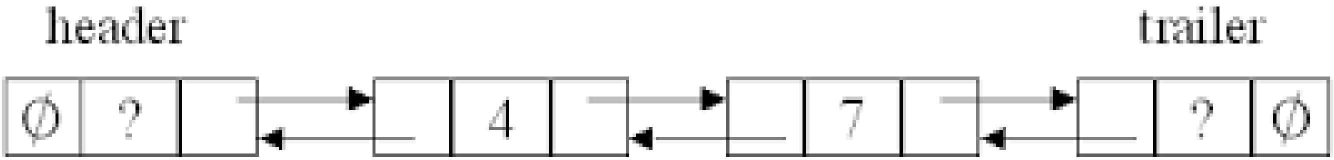
練習題：

- 反序鏈結串列
  - 使用前例，新增一個成員函式，會回傳出反序的鏈結串列。
    - bool Inverse(CLinkedList& ls) (A.inverse(B) B 是A的反向)
- 鏈式堆疊佇列(自由練習)
  - 使用鏈結串列實做堆疊功能。
  - 使用鏈結串列實做佇列功能。
  - 使用鏈結串列實做環狀佇列功能
- 整合add back, add front, insert 成一個，就叫做insert

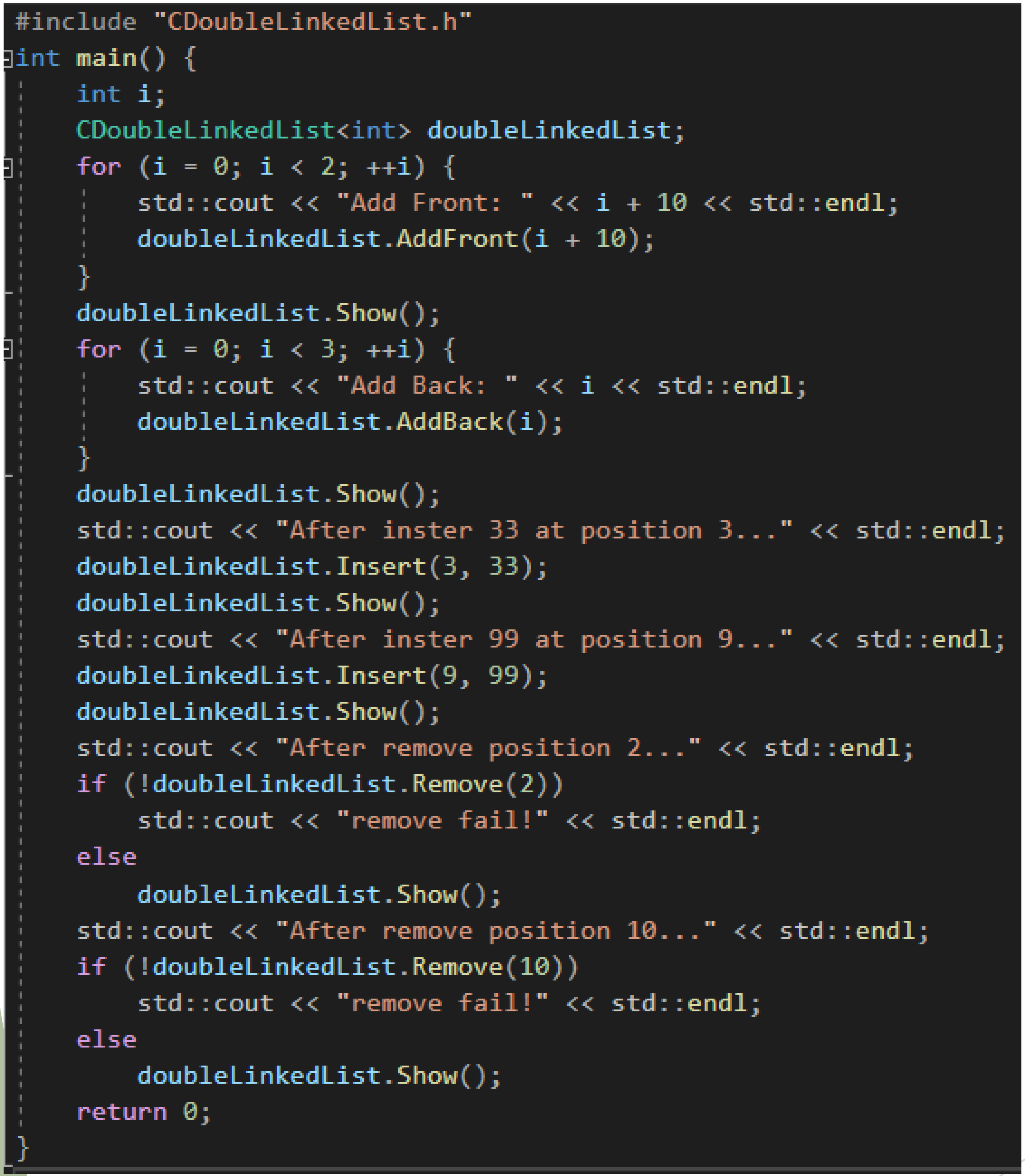
雙向鏈結串列(可以往前走，或往後走)

pointer pointer

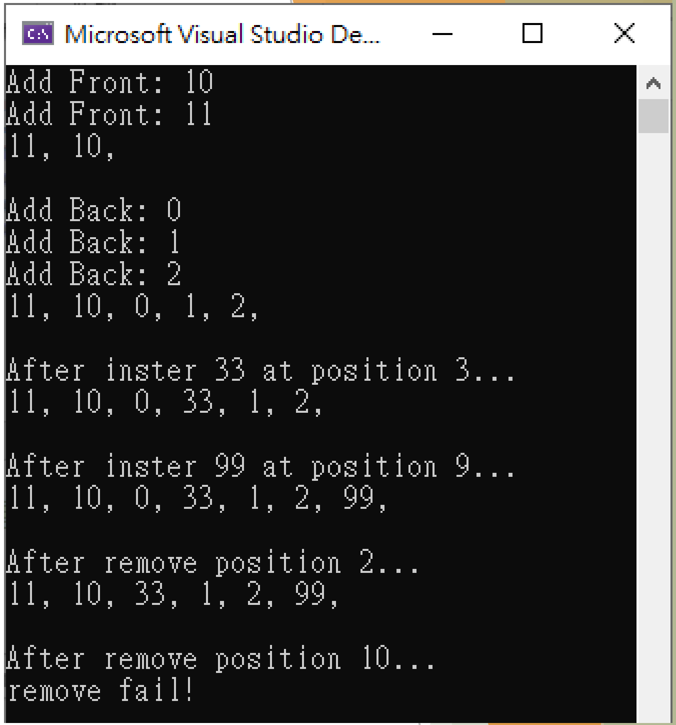




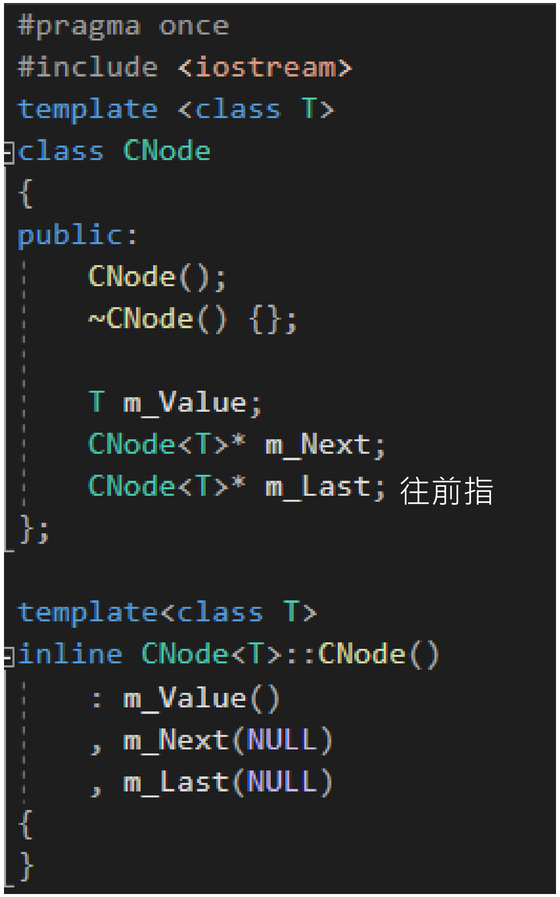
操作多一倍，往後串



```
#include "CDoubleLinkedList.h"
int main() {
    int i;
    CDoubleLinkedList<int> doubleLinkedList;
    for (i = 0; i < 2; ++i) {
        std::cout << "Add Front: " << i + 10 << std::endl;
        doubleLinkedList.AddFront(i + 10);
    }
    doubleLinkedList.Show();
    for (i = 0; i < 3; ++i) {
        std::cout << "Add Back: " << i << std::endl;
        doubleLinkedList.AddBack(i);
    }
    doubleLinkedList.Show();
    std::cout << "After inster 33 at position 3..." << std::endl;
    doubleLinkedList.Insert(3, 33);
    doubleLinkedList.Show();
    std::cout << "After inster 99 at position 9..." << std::endl;
    doubleLinkedList.Insert(9, 99);
    doubleLinkedList.Show();
    std::cout << "After remove position 2..." << std::endl;
    if (!doubleLinkedList.Remove(2))
        std::cout << "remove fail!" << std::endl;
    else
        doubleLinkedList.Show();
    std::cout << "After remove position 10..." << std::endl;
    if (!doubleLinkedList.Remove(10))
        std::cout << "remove fail!" << std::endl;
    else
        doubleLinkedList.Show();
    return 0;
}
```

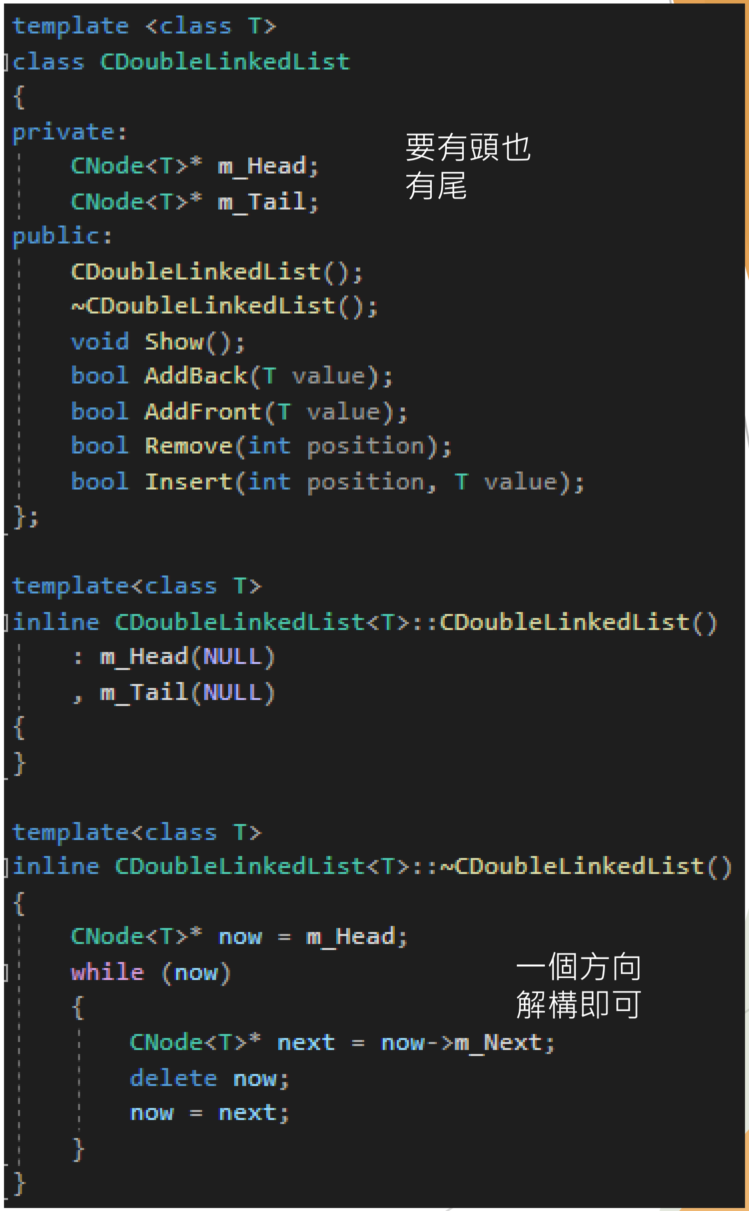


```
Add Front: 10
Add Front: 11
11, 10,

Add Back: 0
Add Back: 1
Add Back: 2
11, 10, 0, 1, 2,

After inster 33 at position 3...
11, 10, 0, 33, 1, 2,

After inster 99 at position 9...
11, 10, 0, 33, 1, 2, 99,

After remove position 2...
11, 10, 33, 1, 2, 99,

After remove position 10...
remove fail!
```



```
#pragma once
#include <iostream>
template <class T>
class CNode
{
public:
    CNode();
    ~CNode() {};

    T m_Value;
    CNode<T>* m_Next;
    CNode<T>* m_Last;
};

template<class T>
inline CNode<T>::CNode()
    : m_Value()
    , m_Next(NULL)
    , m_Last(NULL)
{
}
```

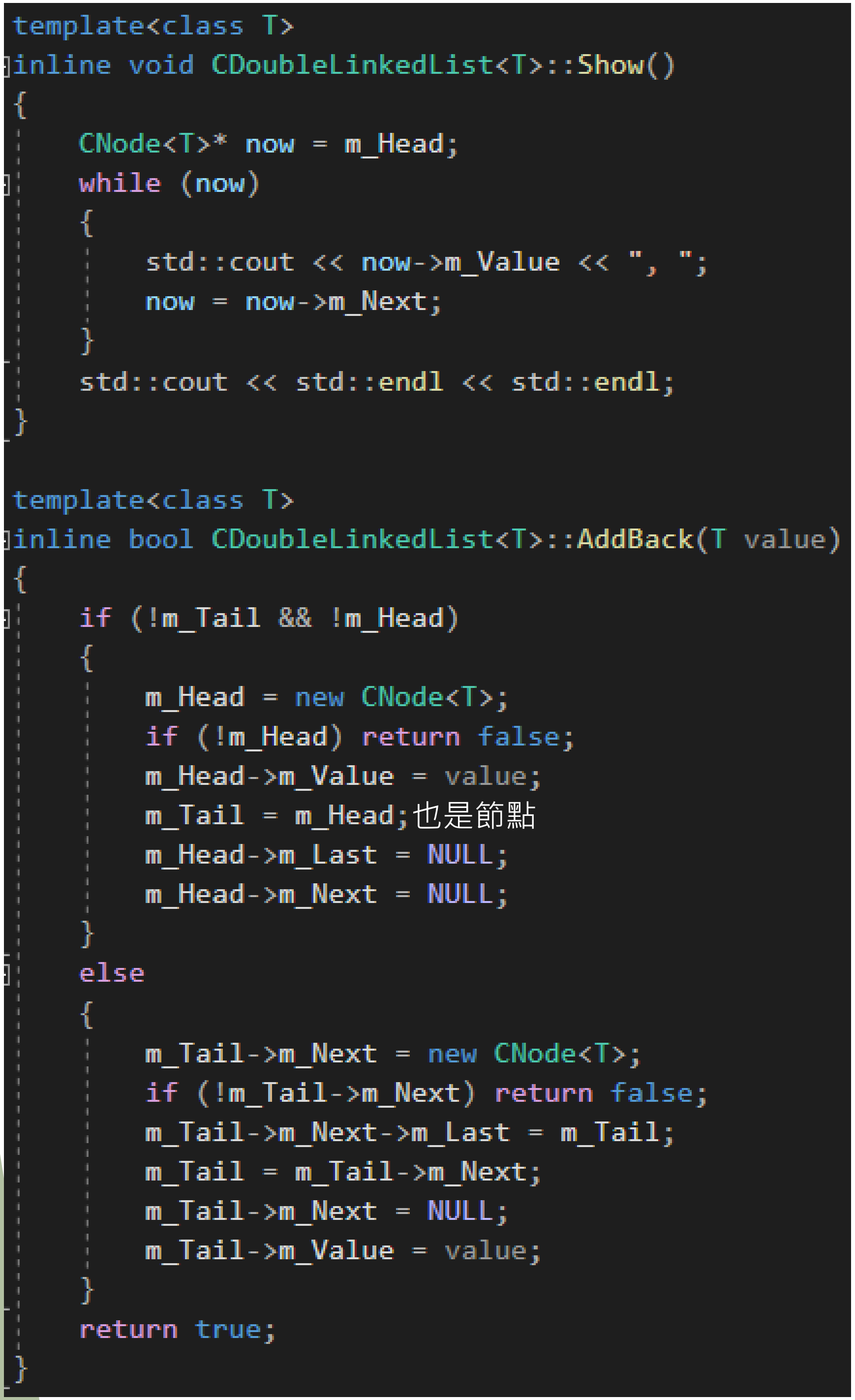
往前指



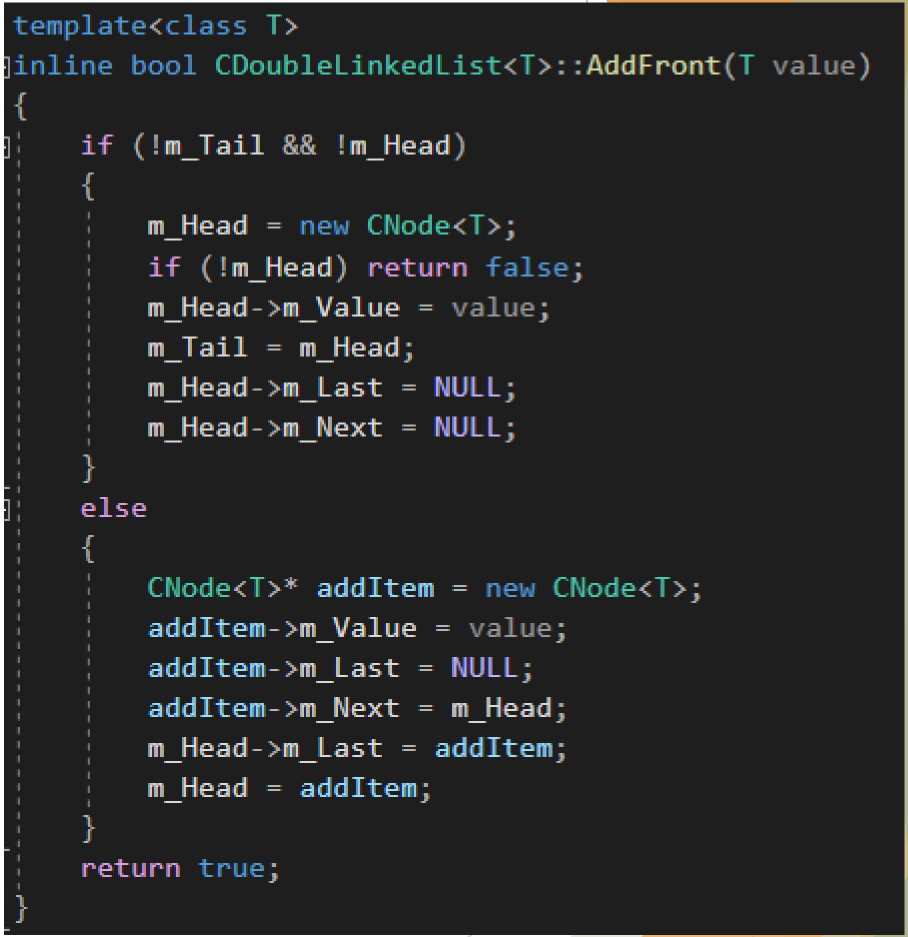
```
template <class T>
class CDoubleLinkedList
{
private:
    CNode<T>* m_Head;
    CNode<T>* m_Tail;
public:
    CDoubleLinkedList();
    ~CDoubleLinkedList();
    void Show();
    bool AddBack(T value);
    bool AddFront(T value);
    bool Remove(int position);
    bool Insert(int position, T value);
};

template<class T>
inline CDoubleLinkedList<T>::CDoubleLinkedList()
    : m_Head(NULL)
    , m_Tail(NULL)
{
}

template<class T>
inline CDoubleLinkedList<T>::~CDoubleLinkedList()
{
    CNode<T>* now = m_Head;
    while (now)
    {
        CNode<T>* next = now->m_Next;
        delete now;
        now = next;
    }
}
```

要有頭也
有尾

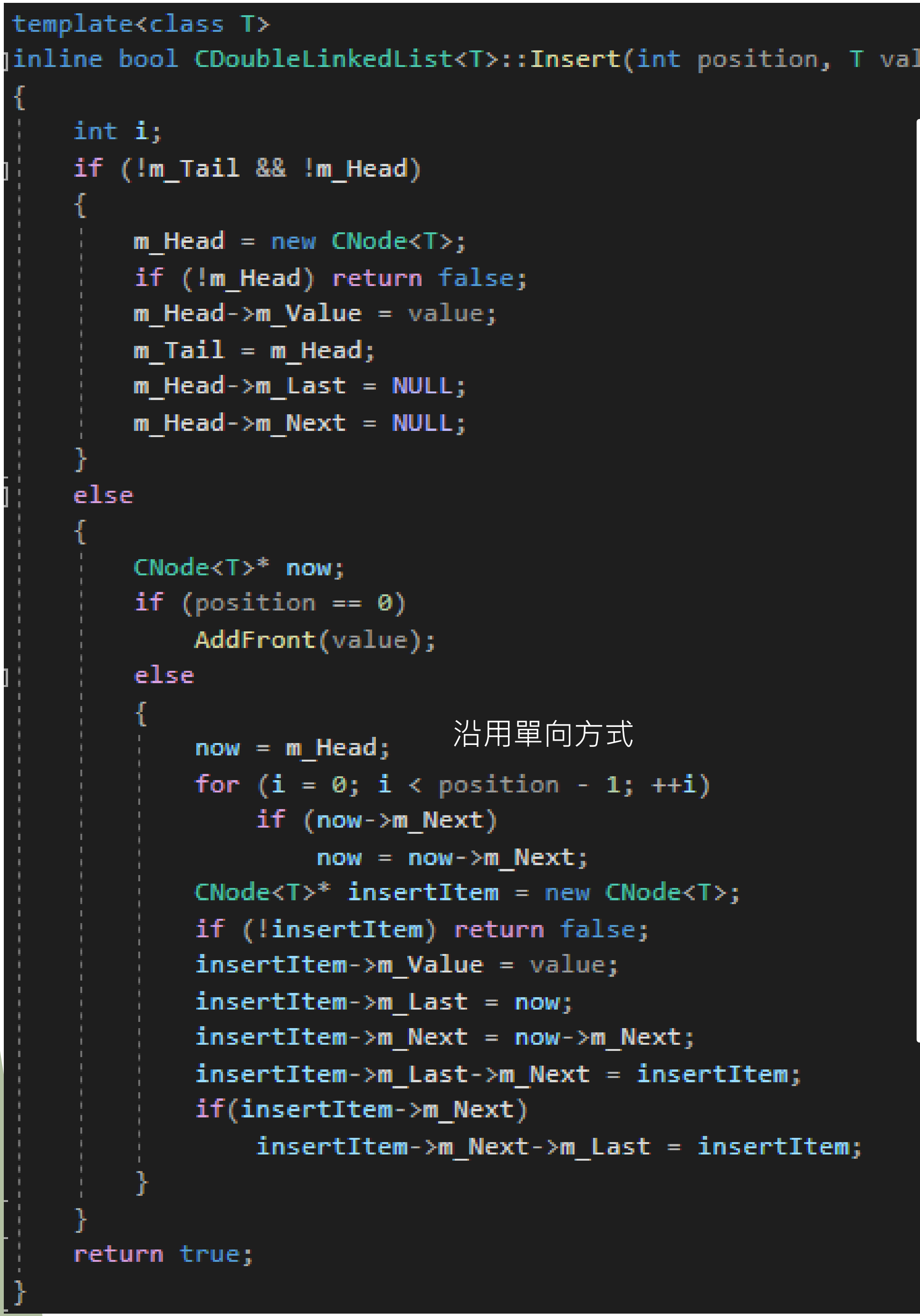
一個方向
解構即可



```
template<class T>
inline void CDoubleLinkedList<T>::Show()
{
    CNode<T>* now = m_Head;
    while (now)
    {
        std::cout << now->m_Value << ", ";
        now = now->m_Next;
    }
    std::cout << std::endl << std::endl;
}

template<class T>
inline bool CDoubleLinkedList<T>::AddBack(T value)
{
    if (!m_Tail && !m_Head)
    {
        m_Head = new CNode<T>;
        if (!m_Head) return false;
        m_Head->m_Value = value;
        m_Tail = m_Head;也是節點
        m_Head->m_Last = NULL;
        m_Head->m_Next = NULL;
    }
    else
    {
        m_Tail->m_Next = new CNode<T>;
        if (!m_Tail->m_Next) return false;
        m_Tail->m_Next->m_Last = m_Tail;
        m_Tail = m_Tail->m_Next;
        m_Tail->m_Next = NULL;
        m_Tail->m_Value = value;
    }
    return true;
}
```

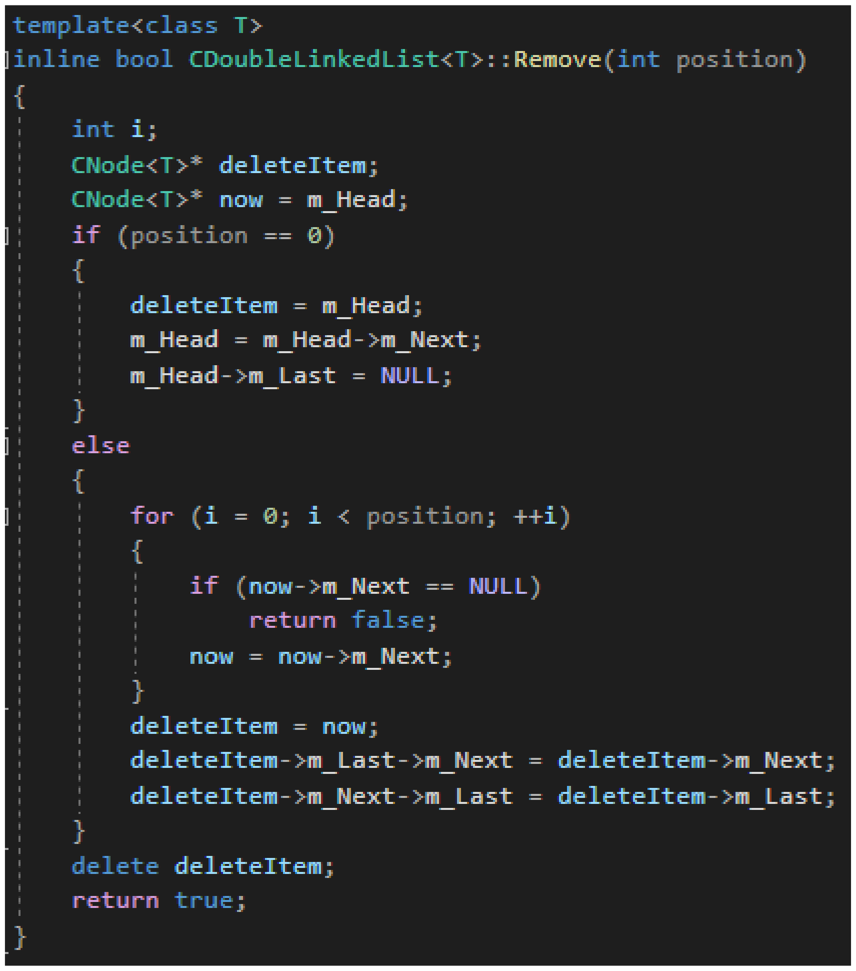


```
template<class T>
inline bool CDoubleLinkedList<T>::AddFront(T value)
{
    if (!m_Tail && !m_Head)
    {
        m_Head = new CNode<T>;
        if (!m_Head) return false;
        m_Head->m_Value = value;
        m_Tail = m_Head;
        m_Head->m_Last = NULL;
        m_Head->m_Next = NULL;
    }
    else
    {
        CNode<T>* addItem = new CNode<T>;
        addItem->m_Value = value;
        addItem->m_Last = NULL;
        addItem->m_Next = m_Head;
        m_Head->m_Last = addItem;
        m_Head = addItem;
    }
    return true;
}
```



```
template<class T>
inline bool CDoubleLinkedList<T>::Insert(int position, T value)
{
    int i;
    if (!m_Tail && !m_Head)
    {
        m_Head = new CNode<T>;
        if (!m_Head) return false;
        m_Head->m_Value = value;
        m_Tail = m_Head;
        m_Head->m_Last = NULL;
        m_Head->m_Next = NULL;
    }
    else
    {
        CNode<T>* now;
        if (position == 0)
            AddFront(value);
        else
        {
            now = m_Head;
            for (i = 0; i < position - 1; ++i)
                if (now->m_Next)
                    now = now->m_Next;
            CNode<T>* insertItem = new CNode<T>;
            if (!insertItem) return false;
            insertItem->m_Value = value;
            insertItem->m_Last = now;
            insertItem->m_Next = now->m_Next;
            insertItem->m_Last->m_Next = insertItem;
            if(insertItem->m_Next)
                insertItem->m_Next->m_Last = insertItem;
        }
    }
    return true;
}
```

沿用單向方式



```
template<class T>
inline bool CDoubleLinkedList<T>::Remove(int position)
{
    int i;
    CNode<T>* deleteItem;
    CNode<T>* now = m_Head;
    if (position == 0)
    {
        deleteItem = m_Head;
        m_Head = m_Head->m_Next;
        m_Head->m_Last = NULL;
    }
    else
    {
        for (i = 0; i < position; ++i)
        {
            if (now->m_Next == NULL)
                return false;
            now = now->m_Next;
        }
        deleteItem = now;
        deleteItem->m_Last->m_Next = deleteItem->m_Next;
        deleteItem->m_Next->m_Last = deleteItem->m_Last;
    }
    delete deleteItem;
    return true;
}
```

作業：環狀鏈結串列

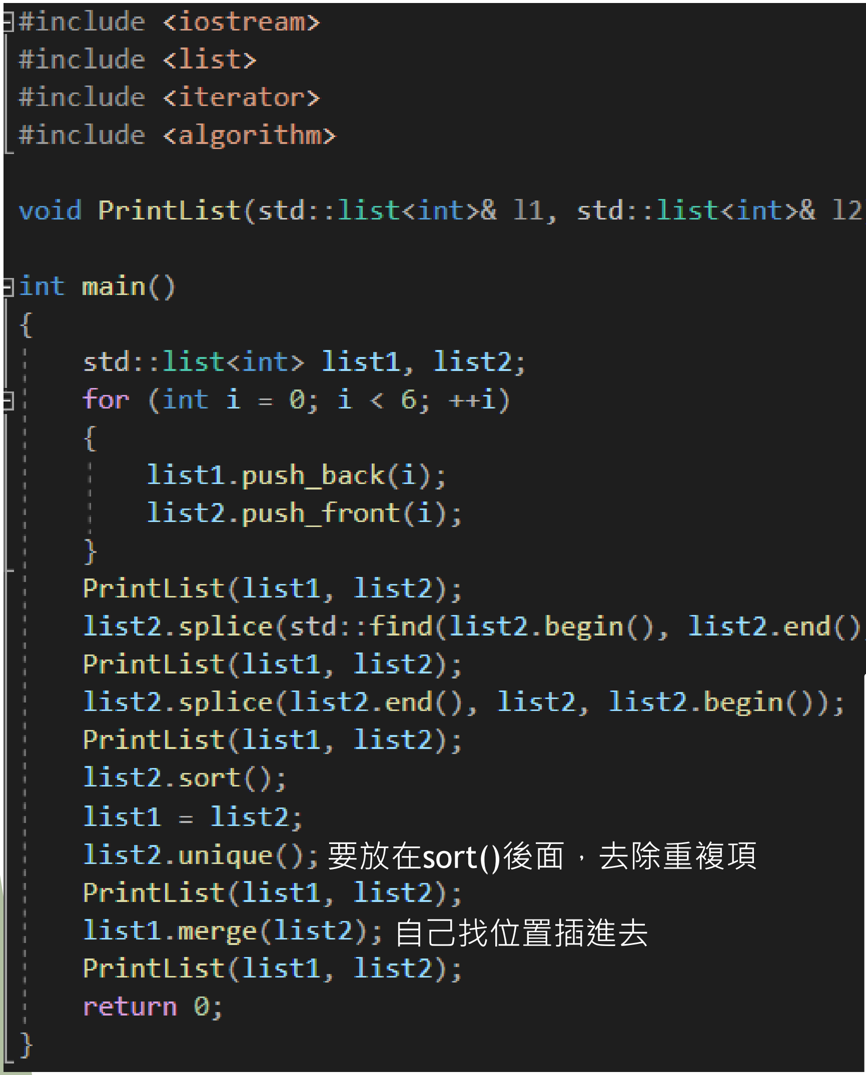
- 修改前例，將其改為雙向環狀鏈結串列
  - 第一個點的往前指向最後一個點
  - 最後一個點往後指指向第一個點
  - Show() 還有 destructor 會有問題，要去解決

C++標準函式庫-list

- 支援泛型，自動擴展長度。
- 在任何位置皆可快速增刪元素。(指標調整)
- 不支援隨機存取，僅能循序存取。
- 須含入list檔來使用list容器。
- 需含入iterator檔來循序存取元素。
- 循序存取時須使用iterator類別。
- 參考資料：http://www.cplusplus.com/reference/list/list/

Python 的 list 是 C++ 中的 stack，
不要跟C++中的 list 搞混了!

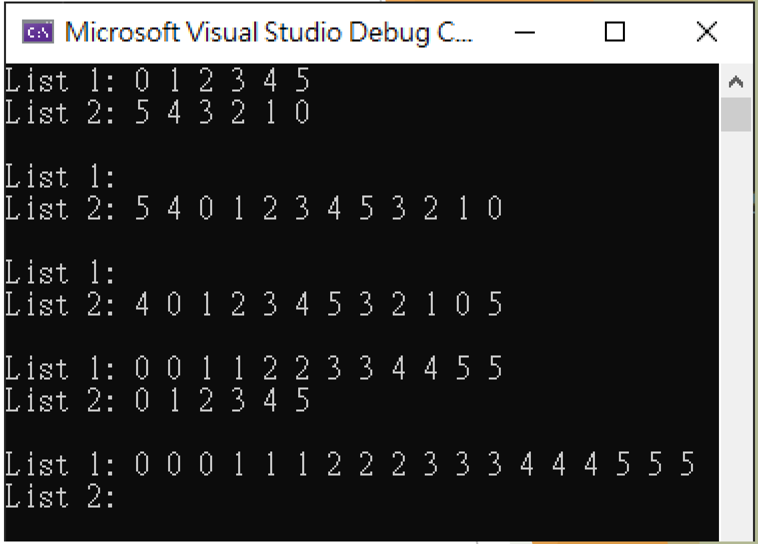
List範例



```cpp
#include <iostream>
#include <list>
#include <iterator>
#include <algorithm>

void PrintList(std::list<int>& l1, std::list<int>& l2);

int main()
{
    std::list<int> list1, list2;
    for (int i = 0; i < 6; ++i)
    {
        list1.push_back(i);
        list2.push_front(i);
    }
    PrintList(list1, list2);
    list2.splice(std::find(list2.begin(), list2.end(), 3), list1);
    PrintList(list1, list2);
    list2.splice(list2.end(), list2, list2.begin());
    PrintList(list1, list2);
    list2.sort();
    list1 = list2;
    list2.unique();
    PrintList(list1, list2);
    list1.merge(list2);
    PrintList(list1, list2);
    return 0;
}
```

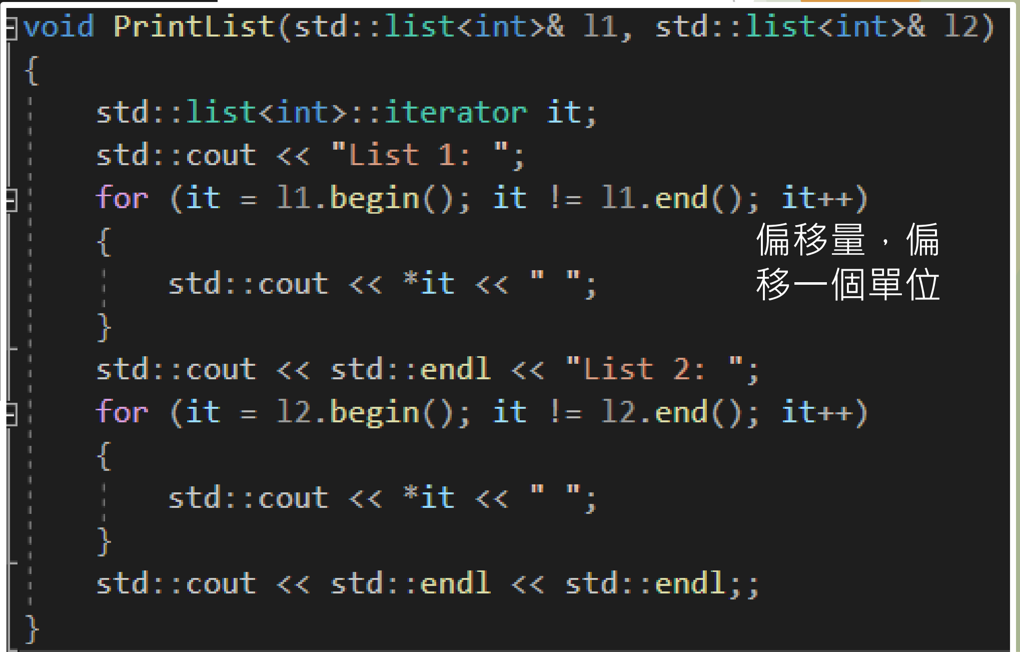
list2.unique(); 要放在sort()後面，去除重複項

list1.merge(list2); 自己找位置插進去



```
List 1: 0 1 2 3 4 5
List 2: 5 4 3 2 1 0

List 1:
List 2: 5 4 0 1 2 3 4 5 3 2 1 0

List 1:
List 2: 4 0 1 2 3 4 5 3 2 1 0 5

List 1: 0 0 1 1 2 2 3 3 4 4 5 5
List 2: 0 1 2 3 4 5

List 1: 0 0 0 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5
List 2:
```

++有overloaded



```cpp
void PrintList(std::list<int>& l1, std::list<int>& l2)
{
    std::list<int>::iterator it;
    std::cout << "List 1: ";
    for (it = l1.begin(); it != l1.end(); it++)
    {
        std::cout << *it << " ";
    }
    std::cout << std::endl << "List 2: ";
    for (it = l2.begin(); it != l2.end(); it++)
    {
        std::cout << *it << " ";
    }
    std::cout << std::endl << std::endl;;
}
```

偏移量，偏移一個單位

想想看 拿時間換空間，拿空間換時間

- 鏈結串列無法隨機存取僅能循序存取，是否有方法降低存取元素所造成的額外成本。
  - (額外指標，存在陣列中，也就是中繼站，直接切入，但又要額外花空間)
- 鏈結串列與長度可增長的堆疊相比，已失去了沒有容量限制的優勢，那為什麼還要使用鏈結串列。
  - (鏈結串列比較不用搬移元素，直接斷開，插入即可)