

資料結構與C++進階班

堆疊

講師：黃銀鵬

E-mail: yinpenghuang@gmail.com

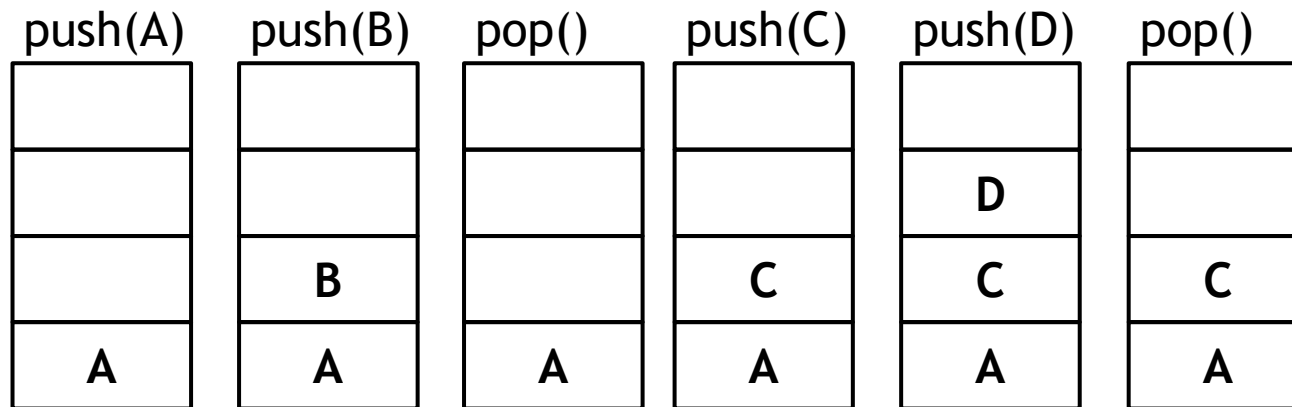
老師教學導向與一般的書
比較不一樣，堆疊與佇列
分成兩個章節教

資結正式開始的章節

基本堆疊

堆疊區(stack) 內就是堆疊

- ▶ 可使用一維陣列結構或鏈結串列實作。(線性)
- ▶ 只允許在陣列尾端操作；最先進入堆疊的元素最後離開堆疊；最後進入堆疊的元素最先離開堆疊。
- ▶ 先進後出規則(LIFO, Last In First Out)。



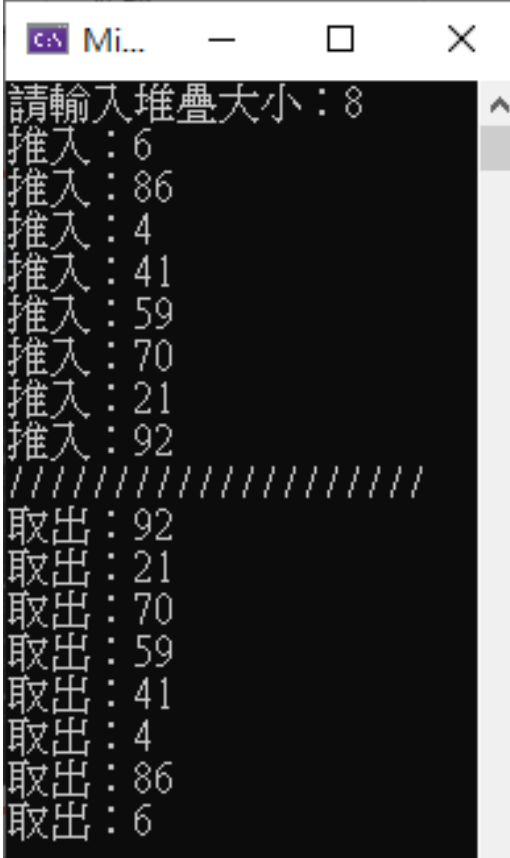
取(pop)離出口最近的值

堆疊實作-陣列

陣列只是擴展

```
#include <iostream>
#include <time.h>
#include "CBaseStack.h"

int main()
{
    int size, i;
    srand(time(NULL));
    CBaseStack<int> baseStack;
    std::cout << "請輸入堆疊大小：" << std::endl;
    std::cin >> size;
    baseStack.SetSize(size); 手抖貼到兩次，導致new兩次
    int val;
    for (i = 0; i < size; ++i)
    {
        val = rand() % 100;
        baseStack.Push(val);
        std::cout << "推入：" << val << std::endl;
    }
    std::cout << "/////////////////" << std::endl;
    for (i = 0; i < size; ++i)
    {
        baseStack.Pop(val);
        std::cout << "取出：" << val << std::endl;
    }
    return 0;
}
```



```
CS Mi...  -  □  ×
請輸入堆疊大小：8
推入：6
推入：86
推入：4
推入：41
推入：59
推入：70
推入：21
推入：92
/////////////////
取出：92
取出：21
取出：70
取出：59
取出：41
取出：4
取出：86
取出：6
```

```

#pragma once
template <class T>
class CBaseStack {
private:
    unsigned int m_Size;
    unsigned int m_End;
    T* m_Element;
public:
    CBaseStack();
    ~CBaseStack();
    bool SetSize(int size);
    bool Push(T value);
    bool Pop(T& val);
};

template<class T>
inline CBaseStack<T>::CBaseStack()
    : m_Size(0)
    , m_End(0)
    , m_Element(NULL)
{
}

template<class T>
inline CBaseStack<T>::~~CBaseStack()
{
    delete[] m_Element;
}

```

永遠指向第一個沒有值的空間

幾乎不會出現其他的return，要讓user知道成功或失敗，bool

```

template<class T>
inline bool CBaseStack<T>::SetSize(int size)
{
    if (m_Element)
        delete[] m_Element;
    m_Element = new T[size];
    if (!m_Element)
        return false;
    m_Size = size;
    return true;
}

template<class T>
inline bool CBaseStack<T>::Push(T value)
{
    m_Element[m_End++] = value;
    return true;
}

template<class T>
inline bool CBaseStack<T>::Pop(T& val)
{
    val = m_Element[--m_End];
    return true;
}

```

曾經有配置過空間，先刪掉舊的

配置失敗return false

先給值再加加，也有可能會失敗，自己去修改

ptr = new int[x]; (ptr刪除還有原本的存取位置)
Delete[] ptr; (因為delete不回傳值)
ptr = NULL; (ptr沒指向合法空間)

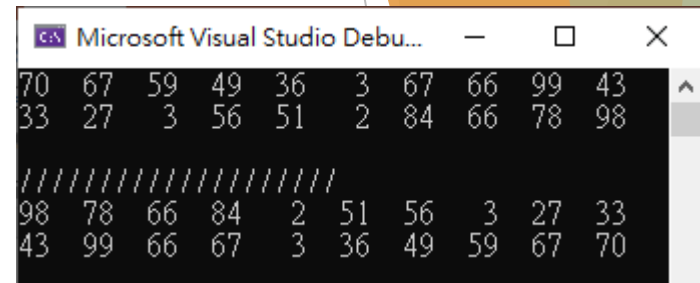
練習：

- ▶ 邊界檢查：(增加return false)
 - ▶ 修改前例，增加邊界的檢查，避免存取出界。
- ▶ 新增與移除中間元素：
 - ▶ 修改前例，增加新增與移除中間元素功能。
 - ▶ `bool Insert(int position, T value);` (滿的是失敗)
 - ▶ `bool Remove(int position);` (空的是失敗)
 - ▶ 前面有空間要擠到前面

例題：自動擴展長度堆疊(會長大的堆疊)

```
#include <iostream>
#include <iomanip>
#include <time.h>
#include "CStack.h"

int main()
{
    int i;
    srand(time(NULL));
    CStack<int> stack;
    int val;
    for (i = 0; i < 20; ++i)
    {
        val = rand() % 100;
        stack.PushBack(val);
        std::cout << std::setw(2) << val << " ";
        if (i % 10 == 9)
            std::cout << std::endl;
    }
    std::cout << std::endl << "/////////////////" << std::endl;
    for (i = 0; i < 20; ++i)
    {
        stack.PopBack(val);
        std::cout << std::setw(2) << val << " ";
        if (i % 10 == 9)
            std::cout << std::endl;
    }
    return 0;
}
```



```
Microsoft Visual Studio Debu...
70 67 59 49 36 3 67 66 99 43
33 27 3 56 51 2 84 66 78 98

//////////////////
98 78 66 84 2 51 56 3 27 33
43 99 66 67 3 36 49 59 67 70
```

資料搬移會花很多時間

```
#pragma once
template <class T>
class CStack {
private:    每次增大多少
    const int m_StepSize;
    T* m_Array;
    unsigned int m_RealSize;
    unsigned int m_Num;
public:
    CStack();
    ~CStack();
    bool PopBack(T& val);
    bool PushBack(T value);
};

template<class T>
inline CStack<T>::CStack()
    : m_StepSize(5)
    , m_Num(0)
{
    下面馬上
    m_RealSize = 5; 就要用到
    m_Array = new T[m_RealSize];
}

template<class T>
inline CStack<T>::~~CStack()
{
    delete[] m_Array;
}
```

```
template<class T>
inline bool CStack<T>::PopBack(T& val)
{
    val = m_Array[--m_Num];
    return true;
}

template<class T>
inline bool CStack<T>::PushBack(T value)
{
    if (m_Num == m_RealSize) 若放滿
    {
        T* array = new T[m_RealSize + m_StepSize];
        if (!array)
            return false; (目的，來源，要複製的位元組數量)
        memcpy(array, m_Array, sizeof(T) * m_RealSize);
        delete[] m_Array;
        m_Array = array;
        m_Array[m_RealSize] = value;
        m_Num = m_RealSize + 1;
        m_RealSize += m_StepSize;
    }
    else if (m_Num < m_RealSize) 若還沒放滿
    {
        m_Array[m_Num] = value;
        m_Num++;
    }
    return true;
}
```

作業：新增與移除中間元素

- ▶ 修改前例，增加新增與移除中間元素功能。
 - ▶ `bool Insert(int position, T value);` (功能被擴展的堆疊)
 - ▶ `bool Remove(int position);`
 - ▶ 清空的時候要`return false;`(沒東西可以remove了)

C++標準函式庫-vector (其實就是template class)

- ▶ 陣列版本(動態記憶體配置)的堆疊，以動態陣列為基礎實作。
- ▶ 支援泛型(廣泛的型別)，自動擴展長度，可隨機存取。
- ▶ 在結尾可以快速增刪元素，在前面或中間增刪元素則相當耗時。(資料搬移)
- ▶ 使用方式接近陣列，經常取代陣列使用。(那為何還要使用陣列，動態配置?)
- ▶ 須含入vector檔。
- ▶ 參考資料：<http://www.cplusplus.com/reference/vector/vector/>
- ▶ C內沒有放在標準函式庫內，針對自己想要的部分開發

剛剛練習的會長大的堆疊，為vector的核心

要對程式抱有疑問，若程式碼跟你想的不一樣

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
```

(強化的陣列)Vector範例

```
int main()
```

(不寫的話，用成長的，每次成長，要花很多時間，這樣可以省掉前期的時間)

```
{
    std::vector<std::string> sentence;
    sentence.reserve(5); // 初始capacity
    sentence.push_back("Hello,");
    sentence.push_back("how");
    sentence.push_back("are");
    sentence.push_back("you");
    sentence.push_back("?");
    for (int i = 0; i < sentence.size(); ++i)
        std::cout << sentence[i] << " ";
    std::cout << std::endl;
    std::cout << "Max_size(): " << sentence.max_size() << std::endl;
    std::cout << "size(): " << sentence.size() << std::endl;
    std::cout << "capacity(): " << sentence.capacity() << std::endl;
    std::cout << std::endl;
    std::swap(sentence[1], sentence[3]);
    sentence.insert(find(sentence.begin(), sentence.end(), "?"), "always");
    sentence.back() = "!";
    for (int i = 0; i < sentence.size(); ++i)
        std::cout << sentence[i] << " ";
    std::cout << std::endl;
    std::cout << "Max_size(): " << sentence.max_size() << std::endl;
    std::cout << "size(): " << sentence.size() << std::endl;
    std::cout << "capacity(): " << sentence.capacity() << std::endl;
    return 0;
}
```

```
Microsoft Vis...
Hello, how are you ?
Max_size(): 153391689 //最大值
size(): 5 // 現在的元素數
capacity(): 5 // 容量

Hello, you are how always !
Max_size(): 153391689 //最大值
size(): 6 // 現在的元素數
capacity(): 7 // 容量
```

每次增加的容量
不一定一樣

並不是要都要用
vector，強大是有
代價，一個元素
所需的使用記憶
體量，遠大於，
使用一般陣列

想想看

- ▶ 堆疊與陣列有什麼特性是相同的，什麼特性是不同的？
- ▶ 什麼時候要用堆疊，什麼時候要用陣列？

用鏈結下去做就沒有隨機存取，

堆疊有LIFO，生活情境：

回到上一步，回到上一頁，返回