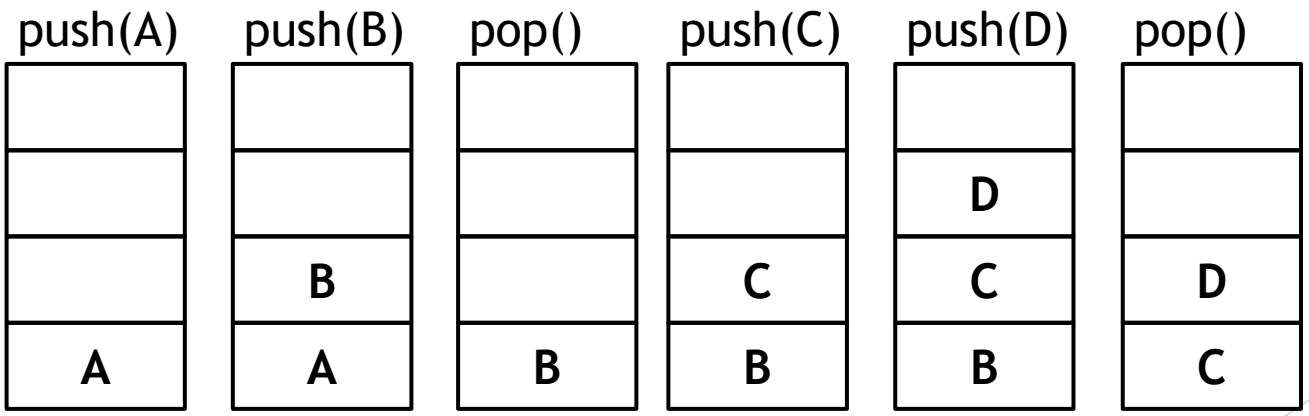
資料結構與C++進階班
佇列

講師：黃銀鵬

E-mail: yinpenghuang@gmail.com

基本佇列

- 可使用一維陣列結構或鏈結串列實作。
- 只允許在陣列尾端加入元素，在陣列頭端移除元素；最先進入堆疊的元素最先離開堆疊；最後進入堆疊的元素最後離開堆疊。(尾進頭出)
- 先進先出規則(FIFO, First In First Out)。(排隊)



佇列實作-陣列

```cpp
#include <iostream>
#include <iomanip>
#include <time.h>
#include "CBaseQueue.h"

int main()
{
    int i, size;
    srand(time(NULL));
    CBaseQueue<int> baseQueue;
    std::cout << "請輸入佇列大小：";
    std::cin >> size;
    baseQueue.SetSize(size);
    int val;
    for (i = 0; i < size; ++i)
    {
        val = rand() % 100;
        baseQueue.Push(val);
        std::cout << std::setw(2) << val << "  " << std::endl;
    }
    std::cout << std::endl << "////////////////////" << std::endl;
    for (i = 0; i < size; ++i)
    {
        baseQueue.Pop(val);
        std::cout << std::setw(2) << val << "  " << std::endl;
    }
    return 0;
}
```

```
請輸入佇列大小：8
20
32
69
64
30
 1
 8
 3

////////////////////
20
32
69
64
30
 1
 8
 3
```

```cpp
#pragma once
template <class T>
class CBaseQueue {
private:
    unsigned int m_Head;
    unsigned int m_Tail;
    unsigned int m_RealNum;
    unsigned int m_Size;
    T* m_Elements;
public:
    CBaseQueue();
    ~CBaseQueue();
    bool SetSize(unsigned int size);
    bool Push(T value);
    bool Pop(T& val);
};

template<class T>
inline CBaseQueue<T>::CBaseQueue()
    : m_Head(0)
    , m_Tail(0)
    , m_RealNum(0)
    , m_Size(0)
    , m_Elements(NULL)
{
}

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

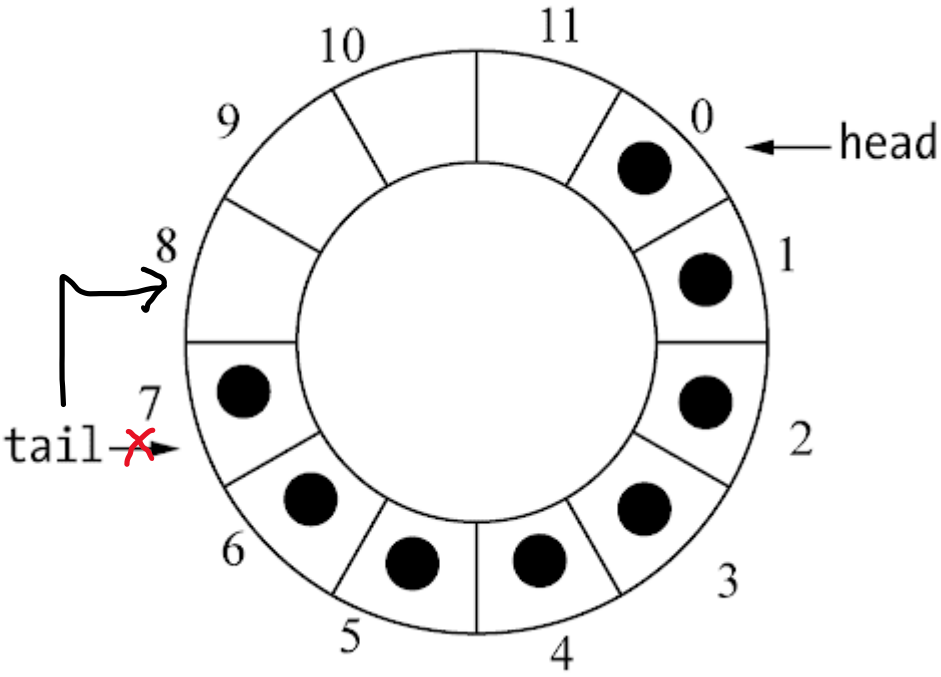
head: 有值的第一個
tail: 無值的第一個
元素數量

```cpp
template<class T>
inline bool CBaseQueue<T>::SetSize(unsigned int size)
{
    if (m_Elements)
        delete[] m_Elements;
    m_Elements = new T[size];
    if (!m_Elements)
        return false;
    m_Size = size;
    return true;
}

template<class T>
inline bool CBaseQueue<T>::Push(T value)
{
    if (m_Tail + 1 > m_Size)
        return false;
    m_Elements[m_Tail++] = value;
    m_RealNum++;
    return true;
}

template<class T>
inline bool CBaseQueue<T>::Pop(T& val)
{
    if (m_RealNum == 0)
        return false;
    val = m_Elements[m_Head++];
    m_RealNum--;
    return true;
}
```

練習

- 新增與移除中間元素
  - 修改基本佇列，增加新增與移除中間元素功能。
  - bool Insert(int position, T value);(前有空格，要往前推)
  - bool Remove(int position); (前有空格，要往前推)
  - Pop產生的空格不理他(pop前面的空間沒用到)，空格到一定的比例再搬動資料
  - 即時性(real time)運算
    - Soft(有彈性，可lag)：網路
    - Hard(嚴格時間規定內回應，不能lag)：看影片，影片本身技術，每秒25張
- Queue用在影片緩衝區buffer，影片就是先進先出。

環形佇列



陣列index

Pop 0, pop 1，尾巴多空間，就不用搬了，但記憶體是線性的，若tail超出了，則回指到0的那個位置，head也超出時，則回指到0的那個位置

```cpp
#include <iostream>
#include <iomanip>
#include <time.h>
#include "CCircularQueue.h"

int main()
{
    int i, size;
    srand(time(NULL));
    CCircularQueue<int> circularQueue;
    std::cout << "請輸入佇列大小：";
    std::cin >> size;
    circularQueue.SetSize(size);
    int val;
    for (i = 0; i < size + 2; ++i)
    {
        val = rand() % 100;
        if (circularQueue.Push(val))
            std::cout << std::setw(2) << val << "  ";
        else
            std::cout << "推入失敗  ";
    }
    std::cout << std::endl << "///////////////" << std::endl;
    for (i = 0; i < size / 2; ++i)
    {
        if (circularQueue.Pop(val))
            std::cout << std::setw(2) << val << "  ";
        else
            std::cout << "取出失敗";
    }
    std::cout << std::endl << "///////////////" << std::endl;
```

Microsoft Visual Studio Debug Console

```
請輸入佇列大小：10
 5  57  22  36   8  77  23  64   2   8  推入失敗  推入失敗
///////////////
 5  57  22  36   8
///////////////
56  23  77  77   2  推入失敗  推入失敗
///////////////
77  23  64   2   8  56  23  77  77   2  取出失敗  取出失敗
```

```cpp
    std::cout << std::endl << "///////////////" << std::endl;
    for (i = 0; i < size / 2 + 2; ++i)
    {
        val = rand() % 100;
        if (circularQueue.Push(val))
            std::cout << std::setw(2) << val << "  ";
        else
            std::cout << "推入失敗  ";
    }
    std::cout << std::endl << "///////////////" << std::endl;
    for (i = 0; i < size + 2; ++i)
    {
        if (circularQueue.Pop(val))
            std::cout << std::setw(2) << val << "  ";
        else
            std::cout << "取出失敗  ";
    }
    return 0;
}
```

```cpp
template<class T>
inline bool CCircularQueue<T>::SetSize(unsigned int size)
{
    if (m_Elements)
        delete[] m_Elements;
    m_Elements = new T[size];
    if (!m_Elements)
        return false;
    m_Size = size;
    return true;
}
```

overload []，在裡面做運算，漿一維的換成二維，表準函示庫內，
堆疊用一維，佇列用二維，函示庫內還是有搬移，環形會比較好

環形通常不長大

```
#pragma once
template <class T>
class CCircularQueue
{
private:
    unsigned int m_Size;      容量
    unsigned int m_Head;
    unsigned int m_Tail;
    unsigned int m_RealNum;   實際存放量
    T* m_Elements;
public:
    CCircularQueue();
    ~CCircularQueue();
    bool SetSize(unsigned int size);
    bool Push(T value);
    bool Pop(T& val);
};

template<class T>
inline CCircularQueue<T>::CCircularQueue()
    : m_Size(0)
    , m_Head(0)
    , m_Tail(0)
    , m_RealNum(0)
    , m_Elements(NULL)
{
}

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

```
template<class T>
inline bool CCircularQueue<T>::Push(T value)
{
    if (m_RealNum == m_Size)
        return false;
    if (m_Tail + 1 > m_Size)
    {
        m_Tail = m_Tail - m_Size;
        m_Elements[m_Tail++] = value;
    }
    else
        m_Elements[m_Tail++] = value;
    m_RealNum++;
    return true;
}

template<class T>
inline bool CCircularQueue<T>::Pop(T& val)
{
    if (m_RealNum == 0)
        return false;
    if (m_Head + 1 > m_Size)
    {
        m_Head = m_Head - m_Size;
        val =  m_Elements[m_Head++];
    }
    else
        val = m_Elements[m_Head++];
    m_RealNum--;
    return true;
}
```

Tail = 0

Head = 0

Delete會自己檢查ptr指向的是不是NULL

作業(雙向皆可進入的佇列)

- 修改前例，將環形佇列加入頭尾皆能存入與取出元素的功能。
  - bool PushBack(T val); (原本的push)
  - bool PushFront(T val);
  - bool PopBack(T& val);
  - bool PopFront(T& val); (原本的pop)

C++標準函式庫-deque

- 陣列版本的佇列，以動態陣列為基礎實作。(動態2維)
- 支援泛型，自動擴展長度，可隨機存取。
- 在頭尾可以快速增刪元素，在中間增刪元素則相當耗時。
- 使用方式接近陣列，經常取代陣列使用。
- 須含入deque檔。
- 參考資料：
  http://www.cplusplus.com/reference/deque/deque/

Deque範例

```
#include <iostream>
#include <string>
#include <deque>
int main()
{
    unsigned int i;
    std::deque<std::string> coll;
    coll.assign(3, std::string("string"));
    coll.push_back("last string");
    coll.push_front("first string");
    for (i = 0; i < coll.size(); ++i)
        std::cout << coll[i] << std::endl;
    std::cout << std::endl;
    coll.pop_front();
    coll.pop_back();
    for (i = 1; i < coll.size(); ++i)
        coll[i] = "another" + coll[i];
    coll.resize(4, "resizeed string");
    for (i = 0; i < coll.size(); ++i)
        std::cout << coll[i] << std::endl;
    return 0;
}
```

(預設空間大小，預設內容)

(多空間，加預設內容)

```
first string
string
string
string
last string

string
anotherstring
anotherstring
resizeed string
```

想想看

- 堆疊與佇列有什麼差異？(前端的產生的方式不同，最明顯差別LIFO vs FIFO)
- 是否有比使用陣列來時做佇列更好的方法。
  - 鏈結串列有好有壞
    - pro:
    - con: 隨機存取沒了，只能循序存取，費時