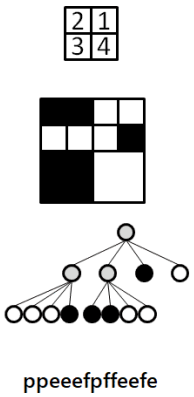
# 國立台灣海洋大學資訊工程系 C++ 程式設計 期中考參考答案

107/04/19(四)

考試時間：**18:40 – 21:00**




右圖中的 Quadtree (四元樹, 四叉樹, Q-tree)是一個在影像處理、電腦圖學、線上遊戲、以及地理資訊系統裡面用途很廣的工具，在影像處理時 Quadtree 可以依照影像各部份資訊量的多寡來調整影像的表示方法，整個區域一片空白或是顏色相同時就可以用一個點來代表整個區域，基本表示方法是將影像分成四個象限，每一部份又可以再分為四個部份...，如果在一個象限中所有像素的顏色都一樣，就可以用單一節點來代表整個區域，下面我們只考慮 32x32 個像素的黑白圖片，右圖一個根節點下面的整個 Quadtree 就代表一張影像，影像的四個象限由四個子節點分別代表，子節點的順序和象限的對應關係如最上圖所示，右圖最下方是這個 Quadtree 例子的 preorder 記錄方法(先寫父節點再寫子節點)，其中 p 代表父節點，e 代表全白色的子節點，f 代表全黑色的子節點。以下我們製作兩個類別一個是以 Quadtree 表示影像的 ImageQT 類別，另一個是以點陣圖表示影像的 ImageBM 類別 (回答時請包含類別宣告和類別實作，並且標註程式需要放在哪一個檔案中)

1. [5] 先假設這兩個類別都有自己的 unitTest()靜態成員函式，請撰寫 main 函式，其內容分別呼叫 ImageBM 類別的 unitTest()函式以及 ImageQT 類別的 unitTest()函式，請引入需要的標頭檔?

**Sol:**

```
// main.cpp

#include "ImageBM.h"
#include "ImageQT.h"
int main() {
	ImageBM::unitTest();
	ImageQT::unitTest();
	return 0;
}
```

2. [10] 請設計 ImageBM 類別，這個類別需要紀錄點陣圖影像中 32x32 個像素的顏色(黑色:1/白色:0)，請運用標準函式庫中的 vector 這個工具類別來設計存放整數的二維陣列 m_bits (提示: 使用兩層的 vector，第一層固定有 32 個元素，每一個元素又是一個 32 個整數的 vector)，並且撰寫建構元函式(constructor)由下列檔案串流中讀入 32 列每列 32 個 0 與 1 的資料:

```
00000000000011111111110000000000
00000000001111111111111000000000
00000000111110000000111000000000
            …
00000000111000000001111000000000
00000001110000000011110000000000
```

請在**類別內**設計一個影像大小的常數 size，數值為32，這個類別滿足下列的單元測試程式

```
ifstream infile1("bunny32x32.txt");
ImageBM bunnyBM(infile1);
```

**Sol:**

```
// ImageBM.h

#pragma once
#include <vector>
#include <iostream>
using namespace std;

class ImageBM {
```

```
public:
    static const int size = 32;
    ImageBM(istream& is);
    static void unitTest();
private:
    vector<vector<int> > m_bits;
};

// ImageBM.cpp

#include "ImageBM.h"
#include <fstream>
using namespace std;

void ImageBM::unitTest() {
    ifstream infile1("bunny32x32.txt");
    ImageBM bunny(infile1);
}

ImageBM::ImageBM(istream& is)
    : m_bits(size) {    // 也可以寫 m_bits(size, vector<int>())
    int i, j;
    char ch;
    for (i=0; i<size; i++)
        for (j=0; j<size; j++) {
            is >> ch;
            m_bits[i].push_back(ch - '0');
        }
}
或是
ImageBM::ImageBM(istream& is)
    : m_bits(size, vector<int>(size)) {
    int i, j;
    char ch;
    for (i=0; i<size; i++)
        for (j=0; j<size; j++) {
            is >> ch;
            m_bits[i][j] = ch - '0';
        }
}
```

3. [10] 請撰寫 ImageBM 類別的拷貝建構元函式 (copy constructor)，請舉例說明拷貝建構元在什麼時候編譯器會使用到，這個類別其實可以不寫拷貝建構元，為什麼?

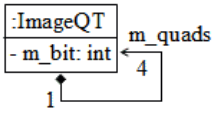
**Sol:**

```
// ImageBM.h

class ImageBM {
    …
public:
    …
    ImageBM(const ImageBM &src);
    …
};

// ImageBM.cpp

ImageBM::ImageBM(const ImageBM &src)
    : m_bits(src.m_bits) {
}
```

這個類別的物件成員是運用vector製作的，vector類別是有設計拷貝建構元的，如果你完全不定義這個類別的拷貝建構元，編譯器幫你合成的建構元是可以使用的，但是你如果自己寫拷貝建構元的話，一定要使用初始化串列初始化m_bits這個vector物件，如果由編譯器幫你合成初始化串列，會完全沒有拷貝的效果

4. [10] 請設計 ImageQT 類別，這個類別需要有一個整數資料成員 m_bit 記錄顏色(1:黑色, 0:白色, -1:整個區域的像素有的是黑色有的是白色)，四個子樹指標陣列的資料成員 m_quads[4] (每一個元素是 ImageQT*型態，類別圖如右)，請撰寫建構元函式由下列檔案串流中讀入整個 Quadtree:

ppeeefpffeefe



滿足下列的單元測試程式

```
ifstream infile2("bunny32x32qt.txt");

ImageQt bunnyQT(infile2);
```

你可以參考一下這段檢查兩個 ImageQT 類別物件是否具有完全相同架構的函式

```
bool ImageQT::equals(const ImageQT &rhs) const {
    if ((m_bit>=0)||(rhs.m_bit>=0))
        return m_bit == rhs.m_bit;
    else // ((m_bit==-1)&&(rhs.m_bit==-1)) {
        for (int i=0; i<4; i++)
            if (!m_quads[i]->equals(*rhs.m_quads[i]))
                return false;
        return true;
    }
}
```

**Sol:**

```
// ImageQT.h

#pragma once
#include <iostream>
using namespace std;
class ImageQT {
public:
    ImageQT(istream &is);
    static void unitTest();
private:
    int m_bit;
    ImageQT *m_quads[4];
};

// ImageQT.cpp

#include "ImageQT.h"
#include <fstream>
using namespace std;

void ImageQT::unitTest() {
    ifstream infile("bunny32x32qt.txt");
    ImageQT bunny(infile);
}

ImageQT::ImageQT(istream &is) {
    int i;
    char c;
    is >> c;
    switch(c) {
    case 'p':
        m_bit = -1;
        for (i=0; i<4; i++)
            m_quads[i] = new ImageQT(is);
        break;
    case 'f':
        m_bit = 1; // black
        for (i=0; i<4; i++)
            m_quads[i] = 0;
        break;
    case 'e':
        m_bit = 0; // white
        for (i=0; i<4; i++)
            m_quads[i] = 0;
    }
}
```

5. [5] 請撰寫 ImageQT 類別的建構元(constructor)函式 ImageQT(int bw)將 m_bit 初始化為傳入的整數 bw，將指標 m_quads[4] 初始化為 0，請運用預設參數的語法將這個建構元設計成預設建構元(default constructor)

**Sol:**

```
// ImageQT.h
```

```
class ImageQT {
public:
    ImageQT(int bw=-1);
    ...
};

// ImageQT.cpp

ImageQT::ImageQT(int bw)
    : m_bit(bw) {
    for (int i=0; i<4; i++)
        m_quads[i] = 0;
}
```

6. [5] 請撰寫 ImageQT 類別的解構元(destructor)函式將以該節點為根節點的整個子樹刪除掉

**Sol:**

```
// ImageQT.h

class ImageQT {
public:
    ~ImageQT();
    ...
};

// ImageQT.cpp

ImageQT::~ImageQT() {
    for (int i=0; i<4; i++)
        delete m_quads[i];
}
```

7. [5] 請撰寫 ImageQT 類別的拷貝建構元(copy constructor)函式，請問這個類別如果不寫拷貝建構元可能會發生什麼問題?

**Sol:**

```
// ImageQT.h

class ImageQT {
public:
    ImageQT(const ImageQT& src);
    ...
};

// ImageQT.cpp

ImageQT::ImageQT(const ImageQT& src) {
    int i;
    m_bit = src.m_bit;
    if (m_bit>=0)
        for (i=0; i<4; i++)
            m_quads[i] = 0;
    else
        for (i=0; i<4; i++)
            m_quads[i] = new ImageQT(*src.m_quads[i]);
}
```

這個類別如果不寫解構元的話，只要發生拷貝的動作，編譯器幫你製作的拷貝建構元只會做shallow copy，所以從第二層以下的節點都不會複製到，一旦原本的物件或是拷貝出來的物件解構時，就會出現dangling reference

8. [10] 請撰寫 ImageQT 類別的設定運算子(assignment operator)函式 operator=()，以及對應的測試程式碼

**Sol:**

```
// ImageQT.h

class ImageQT {
public:
    ImageQT& operator=(const ImageQT& rhs);
    ...
};
```

```
// ImageQT.cpp

ImageQT& ImageQT::operator=(const ImageQT& rhs) {
	int i;
	if (&rhs==this) return *this;
	for (i=0; i<4; i++)
		delete m_quads[i];
	m_bit = rhs.m_bit;
	for (i=0; i<4; i++)
		m_quads[i] = new ImageQT(*rhs.m_quads[i]);
	return *this;
}

void ImageQT::unitTest() {
	ifstream infile1("bunny32x32qt.txt");
	ImageQT bunny1(infile1);
	{
	    ImageQT bunny2 = bunny1;
	    assert(bunny2.equals(bunny1));
	 }
	 bunny1.writeQT(cout); // cause error in case of dangling reference

	ifstream infile2("bunny32x32qt.txt");
	ImageQT bunny3(infile2);
	{
	    ImageQT bunny4;
	    bunny4 = bunny3;
	    assert(bunny4.equals(bunny3));
	 }
	 bunny3.writeQT(cout); // cause error in case of dangling reference
}
```

9. [15] 請設計一個讓 ImageQT 類別的物件能夠轉換為 ImageBM 類別物件的建構元 (請注意要完成這個功能時，因為 ImageQT 類別的資料成員 ImageQT::m_bit 以及 ImageQT::m_quads 存放私有的影像資料，不該讓 ImageBM 類別的成員函式讀取，所以要倒過來思考，當我們需要轉換成點陣圖影像時，可以把空的陣列 ImageBM::m_bits 傳給 ImageQT 類別的成員函式，讓它幫忙填寫影像的內容，例如製作一個 ImageQT::fillBitmap()界面，這個界面還需要指定影像的左上角座標以及影像的邊長，使用起來類似 src.fillBitmap(m_bits, 0, 0, size)，後面這三個參數主要是為了遞迴的函式呼叫設計的)

**Sol:**

```
// ImageBM.h

class ImageBM
{
public:
	...
	ImageBM(const ImageQT& qt);
	...
};

// ImageBM.cpp

ImageBM::ImageBM(const ImageQT& qt)
	: m_bits(size, vector<int>(size, 0))
{
	qt.fillBM(m_bits, 0, 0, size);
}

// ImageQT.h

class ImageQT
{
public:
	...
	void fillBM(vector<vector<int> > &bits, int row, int col, int size) const;
	...
};
```

```
// ImageQT.cpp

void ImageQT::fillBM(vector<vector<int> > &bits, int row, int col, int size) const {
        int i, j;
        if (m_bit>=0) {
                for (i=0; i<size; i++)
                        for (j=0; j<size; j++)
                                bits[row+i][col+j] = m_bit;
        }
        else {
                int offsets[4][2]={{0,size/2},{0,0},{size/2,0},{size/2,size/2}};
                for (i=0; i<4; i++)
                        m_quads[i]->fillBM(bits, row+offsets[i][0], col+offsets[i][1], size/2);
        }
}
```

10. [10] 請替 ImageBM 類別設計 writeBM(ostream&)界面，以便在螢幕或是檔案中輸出如右圖的 32x32 點陣資料 (黑色請輸出 '*' 字元，白色請輸出空白字元)，請運用上題的型態轉換建構元，替 ImageQT 類別也設計一個 writeBM(ostream&)界面

**Sol:**

```
// ImageBM.h

class ImageBM {
public:
        ...
        void writeBM(ostream& os) const;
        ...
};

// ImageBM.cpp

void ImageBM::writeBM(ostream& os) const {
        int i, j;
        for (i=0; i<size; i++) {
                for (j=0; j<size; j++)
                        os << (m_bits[i][j]==1 ? "* " : "  ");
                os << '\n';
        }
}

// ImageQT.h

class ImageQT {
public:
        ...
        void writeBM(ostream& os) const;
        ...
};

// ImageQT.cpp

void ImageQT::writeBM(ostream& os) const {
        ImageBM bm(*this);
        bm.writeBM(os);
}
```

11. [5] 請替 ImageBM 類別設計一個 int countBlackPixels() 界面，計算影像中有幾個黑色的像素

**Sol:**

```
// ImageBM.h

class ImageBM {
public:
        ...
        int countBlackPixels() const;
        ...
};

// ImageBM.cpp

int ImageBM::countBlackPixels() const {
```

```
    int i, j, count=0;
    for (i=0; i<size; i++)
        for (j=0; j<size; j++)
            count += m_bits[i][j];
    return count;
}
```

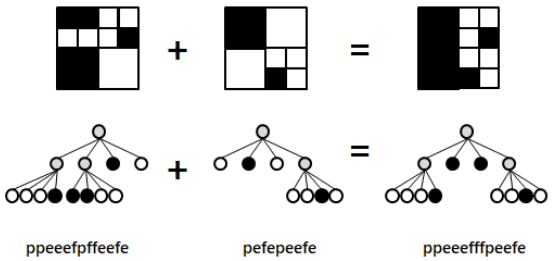
12. [10] 請替 ImageQT 類別設計一個 int countBlackPixels() 界面，計算影像中有幾個黑色的像素

**Sol:**

```
// ImageQT.h

class ImageQT {
public:
    ...
    int countBlackPixels(int level=0) const;
    ...
};

// ImageQT.cpp

int ImageQT::countBlackPixels(int level) const {
    int count = 0, nPixels=1024, nLevels=level+1;
    while (--nLevels) nPixels /= 4;
    if (m_bit>=0)
        return (m_bit == 1)*nPixels;
    for (int i=0; i<4; i++)
        count += m_quads[i]->countBlackPixels(level+1);
    return count;
}
```

13. [15] 如右圖兩張 ImageQT 類別的影像可以做一個「相加」的運算，兩張影像對應的像素有一張是黑色或是兩張都是黑色，相加的結果就是黑色，請替 ImageQT 類別撰寫一個 void add(const ImageQT&)界面來完成相加的動作，函式中修改接受訊息的物件，請注意相加的時候有可能得到四個子樹都是全黑或是全白的情況，此時需要把四個子樹都刪除掉來簡化這個 Quadtree

**Sol:**




```
// ImageQT.h

class ImageQT {
public:
    ...
    void add(const ImageQT& rhs);
    ...
};

// ImageQT.cpp

void ImageQT::add(const ImageQT& rhs) {
    int i, b, w;
    if ((m_bit>=0)&&(rhs.m_bit>=0))
        m_bit = (m_bit + rhs.m_bit) >= 1;
    else if ((m_bit==-1)&&(rhs.m_bit==-1)) {
        for (b=w=i=0; i<4; i++) {
            m_quads[i]->add(*rhs.m_quads[i]);
            b += m_quads[i]->m_bit==1;
            w += m_quads[i]->m_bit==0;
        }
        if ((b==4) || (w==4)) {
            m_bit = b==4;
            for (i=0; i<4; i++) {
                delete m_quads[i];
                m_quads[i] = 0;
            }
        }
    }
    else if ((m_bit==0)&&(rhs.m_bit==-1)){
        m_bit = -1;
        for (i=0; i<4; i++)
            m_quads[i] = new ImageQT(*rhs.m_quads[i]);
    }
```

```
        else if ((m_bit==-1)&&(rhs.m_bit==1)) {
                m_bit = 1;
                for (i=0; i<4; i++) {
                        delete m_quads[i];
                        m_quads[i] = 0;
                }
        }
}
```