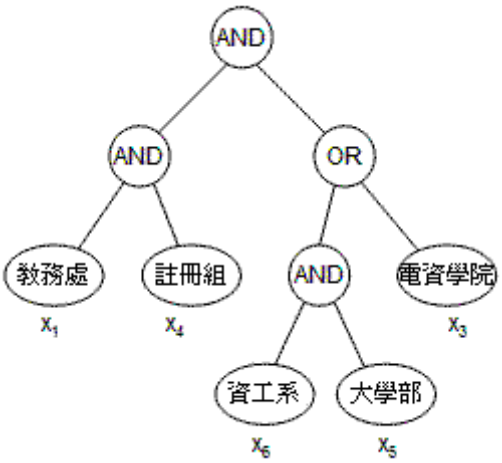
國立台灣海洋大學資訊工程系 C++ 程式設計 期中考試題

姓名：_____學號：_____ 103/04/15

考試時間：**09:30 – 12:00**

試題敘述蠻多的，看清楚題目問什麼，針對重點回答，總分有 **110**，請看清楚每一題所佔的分數再回答

考試規則：1. 不可以翻閱參考書、作業及程式
2. 不可以使用任何形式的電腦 (包含手機、計算機、相機以及其它可運算或是連線的電子器材)
3. 請勿左顧右盼、請勿交談、請勿交換任何資料、試卷題目有任何疑問請舉手發問 (看不懂題目不見得是你的問題，有可能是中英文名詞的問題)、最重要的是隔壁的答案可能比你的還差，白卷通常比錯得和隔壁一模一樣要好
4. 提早繳卷同學請直接離開教室，請勿逗留喧嘩
5. 違反上述任何一點之同學一律送請校方處理
6. 繳卷時請繳交***簽名過之試題卷及答案卷***



在一個資料管理系統中，需要以布林運算式來控管資料的存取，例如資訊工程學系同學的成績資料允許由分配到資訊工程學系大學部，或是分配到電資學院的教務處註冊組職員來存取，這個規則如右圖所示，也可以表示為如下的布林運算式：(教務處 AND 註冊組) AND ((資工系 AND 大學部) OR 電資學院)，每一個職員有其對應的屬性，例如: {教務處(x_1), 註冊組(x_4), 資工系(x_6), 大學部(x_5)}，如果這些屬性滿足某一機密資料對應的布林運算式，系統就允許該職員存取此機密資料。

以下請撰寫一個 BFormula 的類別來代表右圖控管資料存取的運算式 $f(\cdot)=(x_1 \text{ AND } x_4) \text{ AND } ((x_6 \text{ AND } x_5) \text{ OR } x_3)$，另外也請撰寫一個 Attribute 的類別來代表某一使用者具有的屬性集合 $A=\{x_1, x_4, x_6, x_5\}$，亦即 x_1=true, x_4=true, x_6=true, x_5=true, 其它不在 A 中的屬性皆為 false, 以此例而言 f(A) 為 true。

請依照下列的要求完成這兩個類別(BFormula.h, BFormula.cpp, Attribute.h, Attribute.cpp), 以及 main.cpp 的設計, (以下問題回答時請標註程式需要放在哪一個檔案中)

1. [5] 這兩個類別都有自己的 unitTest() static 成員函式，請撰寫 main.cpp, 其內容分別呼叫 Attribute 類別的 unitTest()函式以及 BFormula 類別的 unitTest()函式，請引入需要的標頭檔?

Sol:

```
// main.cpp
#include "BFormula.h"
#include "Attribute.h"
#include <cstdlib>

void main()
{
    Attribute::unitTest();
    BFormula::unitTest();
    system("pause");
}
```

2. [10] 在這個程式中我們只處理最多 100 個屬性: $x_0, x_1, \ldots, x_{99}$, 我們以整數 0, 1, 2, …, 99 來代表這些屬性, 我們運用一個陣列來初始化 Attribute 物件如下:

```
int assign1[]={1,3,5,7,9};
int nassign1 = sizeof(assign1) / sizeof(int);
Attribute a1(assign1, nassign1);
```

Attribute 類別中以一個整數的 vector 物件來存放這些屬性，並且以一個整數來存放屬性的個數，請定義這個類別，設計需要的資料成員，並且撰寫符合上述用法的建構元 (constructor) 函式

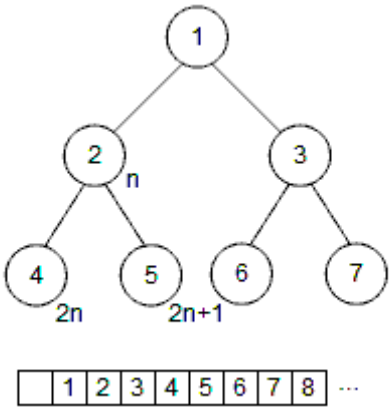
Sol:

```
// attribute.h
#pragma once
#include <vector>
using namespace std;

class Attribute
{
public:
	Attribute(int trueVariables[], int nVariables);
private:
	vector<int> m_trueVariables;
	int m_nTrueVariables;
};

// attribute.cpp
#include "Attribute.h"

Attribute::Attribute(int trueVariables[], int nTrueVariables)
	: m_nTrueVariables(nTrueVariables)
{
	for (int i=0; i<m_nTrueVariables; i++)
		m_trueVariables.push_back(trueVariables[i]);
}
```

3. [5] 請撰寫 Attribute 類別的拷貝建構元函式 (copy constructor)，請問這個類別如果不寫拷貝建構元會有問題嗎?

Sol:

```
// attribute.h
…
class Attribute
{
public:
	Attribute(const Attribute &src);
	…
};

// attribute.cpp
Attribute::Attribute(const Attribute &src)
	: m_nTrueVariables(src.m_nTrueVariables),
	  m_trueVariables(src.m_trueVariables)
{
}
```



因為這個類別沒有自己配置記憶體，其實可以不寫拷貝建構元，上面這個拷貝建構元和 compiler 自己幫你產生的建構元是一模一樣的，請注意如果你自己寫的話，一定要在初始化串列裡串接 vector 類別的拷貝建構元

4. [10] 一個由 AND/OR 組成的布林運算式可以用一個二元樹(Binary Tree)來表示，如上圖一個完整(complete)二元樹可以用陣列來表示，所以 BFormula 這個類別中請**動態配置一個整數陣列**來記錄一個布林運算式，其中陣列的第一個元素不要使用，第二個元素存放根節點的運算子的代碼(AND 運算子請用整數 999 表示, OR 運算子請用整數 998 表示)，陣列的大小也需要有一個整數資料成員來記錄，葉節點為 0~99 的屬性，完整二元樹中以整數 -1 填充不需要的葉節點，例如上圖中布林運算式可以用陣列內資料表示如下

int formula1[] = {-1, 999, 999, 998, 1, 4, 999, 3, -1, -1, -1, -1, 6, 5};

int nformula1 = sizeof(formula1)/sizeof(int);

BFormula f1(formula1, nformula1);

請定義這個類別，設計需要的資料成員，並且撰寫符合上述用法的建構元 (constructor) 函式

Sol:

```
// BFormula.h
#pragma once

class BFormula
{
public:
	BFormula(int formula[], int nformula);
private:
	int *m_formulaTree;
	int m_nNodes;
};

// BFormula.cpp
BFormula::BFormula(int formula[], int nformula)
	: m_nNodes(nformula), m_formulaTree(new int[nformula])
{
	for (int i=0; i<nformula; i++)
		m_formulaTree[i] = formula[i];
}
```

5. [5] 請撰寫 BFormula 類別需要的拷貝建構元函式 (copy constructor)，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:

```
// BFormula.h
class BFormula
{
public:
	BFormula(const BFormula & src);
…
};

// BFormula.cpp
BFormula::BFormula(const BFormula &src)
	: m_nNodes(src.m_nNodes),
	  m_formulaTree(new int[src.m_nNodes])
{
	for (int i=0; i<src.m_nNodes; i++)
		m_formulaTree[i] = src.m_formulaTree[i];
}
```

因為這個類別自己配置且管理記憶體，一定要寫拷貝建構元，否則在拷貝的物件解構時會造成 dangling reference 的記憶體存取問題

6. [5] 請撰寫 BFormula 類別需要的解構元 (destructor) 函式

Sol:

```
// BFormula.h
class BFormula
{
public:
	virtual ~BFormula(void);
…
};

// BFormula.cpp
BFormula::~BFormula(void)
{
	delete[] m_formulaTree;
```

```
}
```

7. [10] 請撰寫 BFormula 類別需要的設定運算子 (assignment operator) 函式 operator=()

Sol:

```
// BFormula.h
class BFormula
{
public:
    BFormula& operator=(BFormula& rhs);
…
};

// BFormula.cpp
BFormula& BFormula::operator=(BFormula& rhs)
{
    if (this == &rhs)
        return *this;
    delete[] m_formulaTree;
    m_nNodes = rhs.m_nNodes;
    m_formulaTree = new int[rhs.m_nNodes];
    for (int i=0; i<rhs.m_nNodes; i++)
        m_formulaTree[i] = rhs.m_formulaTree[i];
    return *this;
}
```

8. [5] 請替 Attribute 類別撰寫一個 contains() 成員函式，傳入一個整數型態的屬性值，如果傳入的屬性值包含於物件代表的屬性集合中則回傳 true，否則回傳 false (請運用 vector 類別的 iterator 或是 const_iterator 來撰寫)

Sol:

```
// Attribute.h
class Attribute
{
public:
    bool contains(const int var) const;
};

// Attribute.cpp
bool Attribute::contains(const int var) const
{
    vector<int>::const_iterator iter;
    for (iter=m_trueVariables.begin(); iter<m_trueVariables.end(); iter++)
        if (var == *iter)
            return true;
    return false;
}
```

9. [20] 請替 BFormula 類別撰寫一個 evaluate 成員函式，傳入一個 Attribute 物件的參考，計算是否能夠滿足這個布林運算式 (由於這個函式不會修改 BFormula 類別的物件，請使用 const 來描述這個成員函式，另外這個函式也不會修改傳入的 Attribute 物件，請使用 const 來描述這個參數，請撰寫 private 的**遞迴**函式來處理此功能，請運用 Attribute::contains()成員函式)

Sol:

```
// BFormula.h
#include "Attribute.h"

class BFormula
{
public:
    bool evaluate(const Attribute& assign) const;
private:
    bool evaluate(const int formula[], const int node, const Attribute& assign) const;
```

```
};

// BFormula.cpp
bool BFormula::evaluate(const Attribute& assign) const
{
	if (m_nNodes>1)
		return evaluate(m_formulaTree, 1, assign);
	else
		return false;
}

bool BFormula::evaluate(const int formula[], const int node, const Attribute& assign) const
{
	if (formula[node] == 999) // AND
		return evaluate(formula, 2*node, assign) &&
			evaluate(formula, 2*node+1, assign);
	else if (formula[node] == 998) // OR
		return evaluate(formula, 2*node, assign) ||
			evaluate(formula, 2*node+1, assign);
	else if ((formula[node]>=0)&&(formula[node]<=MaxVar))
		return assign.contains(formula[node]);
	return false;
}
```

如果你用迴圈寫迭代式的程式，雖然不是題目要求的，程式範例如下

```
// BFormula.h
#include "Attribute.h"

class BFormula
{
public:
	bool evaluateIterative(const Attribute& assign) const;
};

// BFormula.cpp
bool BFormula::evaluateIterative(const Attribute& assign) const
{
	int i, result, *formula = new int[m_nNodes];
	for (i=0; i<m_nNodes; i++)
		formula[i] = m_formulaTree[i];
	for (i=m_nNodes-1; i>0; i--)
	{
		if ((formula[i]>=0)&&(formula[i]<=MaxVar))
			formula[i] = assign.contains(formula[i]); // 1: true, 0: false;
		else if (formula[i] == 999) // AND
			formula[i] = formula[2*i] && formula[2*i+1];
		else if (formula[i] == 998) // OR
			formula[i] = formula[2*i] || formula[2*i+1];
	}
	result = formula[1];
	delete[] formula;
	return result;
}
```

10. [10] 在 BFormula 類別的單元測試函式中，我們希望測試拷貝建構元，設定運算子，並且以 assert 敘述確定屬性集合{1,4,3,8}或是{1,2,4,5,6,8}都可以滿足題 4 的布林運算式 f1，屬性集合{1,3,5}無法滿足布林運算式 f1，請撰寫 unitTest() 函式

Sol:

```
// BFormula.h
class BFormula
{
public:
	static void unitTest();
```

```
…
};

// BFormula.cpp
#include <cassert>
void BFormula::unitTest()
{
    // (x1 and x4) and ((x6 and x5) or x3)       999:AND, 998:OR, -1:NIL, variables:1~100
    int formula1[] = {-1, 999, 999, 998, 1, 4, 999, 3, -1, -1, -1, -1, 6, 5};
    int nformula1 = sizeof(formula1)/sizeof(int);
    BFormula f1(formula1, nformula1);
    BFormula f2 = f1; // 測試拷貝建構元
    BFormula f3; // 需要 default constructor
    f3 = f2; // 測試 assignment operator

    int assign1[] = {1, 4, 3, 8};    // x1=x4=x3=x8=true, all other variables are false
    int nassign1 = sizeof(assign1)/sizeof(int);
    Attribute a1(assign1, nassign1);

    int assign2[] = {1, 2, 4, 5, 6, 8};    // x1=x2=x4=x5=x6=x8=true, all other variables are false
    int nassign2 = sizeof(assign2)/sizeof(int);
    Attribute a2(assign2, nassign2);

    int assign3[] = {1, 3, 5};    // x1=x3=x5=true, all other variables are false
    int nassign3 = sizeof(assign3)/sizeof(int);
    Attribute a3(assign3, nassign3);

    assert(f1.evaluate(a1)); assert(f1.evaluate(a2)); assert(!f1.evaluate(a3));
    assert(f2.evaluate(a1)); assert(f2.evaluate(a2)); assert(!f2.evaluate(a3));
    assert(f3.evaluate(a1)); assert(f3.evaluate(a2)); assert(!f3.evaluate(a3));
}
```

11. [5] 請替 Attribute 類別增加 Union()成員函式，傳入第二個 Attribute 類別物件的參考，計算兩個屬性集合的聯集並且修改物件本身，不需回傳任何數值

Sol:

```
// Attribute.h
class Attribute
{
public:
    void Union(const Attribute &rhs);
};

// Attribute.cpp
void Attribute::Union(const Attribute &rhs)
{
    for (int i=0; i<rhs.m_nTrueVariables; i++)
        if (contains(rhs.m_trueVariables[i]))
            continue;
        else
        {
            m_trueVariables.push_back(rhs.m_trueVariables[i]);
            m_nTrueVariables++;
        }
}
```

12. [5] 請替 Attribute 類別增加 Intersect()成員函式，傳入第二個 Attribute 類別物件的參考，計算兩個屬性集合的交集並且修改物件本身，不需回傳任何數值

Sol:

```
// Attribute.h
class Attribute
{
public:
```

```
    void Intersect(const Attribute &rhs);
};

// Attribute.cpp
void Attribute::Intersect(const Attribute &rhs)
{
    int nNewVariables = 0;
    vector<int> newVariables;
    for (int i=0; i<rhs.m_nTrueVariables; i++)
        if (contains(rhs.m_trueVariables[i]))
        {
            newVariables.push_back(rhs.m_trueVariables[i]);
            nNewVariables++;
        }
    m_trueVariables = newVariables;
    m_nTrueVariables = nNewVariables;
}
```

13. [5] 請替 Attribute 類別增加 equal()成員函式，傳入第二個 Attribute 類別物件的參考，運用 Attribute::contains()來測試兩個 Attribute 物件所代表的屬性集合是否完全一樣，回傳 bool 型態的結果

Sol:

```
// Attribute.h
class Attribute
{
public:
    bool equal(Attribute &rhs) const;
};

// Attribute.cpp
bool Attribute::equal(Attribute &rhs) const
{
    if (m_nTrueVariables != rhs.m_nTrueVariables)
        return false;
    vector<int>::const_iterator iter;
    for (iter=m_trueVariables.begin(); iter<m_trueVariables.end(); iter++)
        if (!rhs.contains(*iter))
            return false;
    return true;
}
```

14. [10] 在 Attribute 類別的單元測試函式中，請設計資料並且運用 assert 測試拷貝建構元, Union() 以及 Intersect()成員函式的運作是否正確，請撰寫 unitTest() 函式

Sol:

```
// Attribute.h
class Attribute
{
public:
    static void unitTest(void);
};

// Attribute.cpp
#include <cassert>
void Attribute::unitTest(void)
{
    int assign1[]={1,3,5,7,9};
    int nassign1 = sizeof(assign1) / sizeof(int);
    int assign2[]={2,3,4,7,10};
    int nassign2 = sizeof(assign2) / sizeof(int);
    int assign3[]={1,2,3,4,5,7,9,10};
```

```
    int nassign3 = sizeof(assign3) / sizeof(int);
    int assign4[]={3,7};
    int nassign4 = sizeof(assign4) / sizeof(int);
    int i;
    Attribute a1(assign1, nassign1);
    Attribute a2(assign2, nassign2);
    Attribute a3(assign3, nassign3);
    Attribute a4(assign4, nassign4);

    Attribute a5(a1); // 測試拷貝建構元
    for (i=0; i<nassign1; i++)
    {
        assert(a1.contains(assign1[i]));
        assert(a5.contains(assign1[i]));
    }

    a5.Union(a1); // 測試相同的屬性集合的聯集
    assert(a1.equal(a5));

    a5.Intersect(a1); // 測試相同的屬性集合的交集
    assert(a5.equal(a1));

    a5.Union(a2); // {1,3,5,7,9} ∪ {2,3,4,7,10} = {1,2,3,4,5,7,9,10} 測試聯集
    assert(a5.equal(a3));

    a1.Intersect(a2); // {1,3,5,7,9} ∩ {2,3,4,7,10} = {3,7} 測試交集
    assert(a4.equal(a1));
}
```