# 第八章 - 運算子多載
# Operator Overloading

**8.1** 簡介
**8.2** 運算子多載基本原理
**8.3** 運算子多載的限制
**8.4** 運算子函式-成員函式與非成員函式
**8.5** 多載<<與>>運算子
**8.6** 多載單運算元運算子(Unary Operators)
**8.7** 多載雙運算元運算子(Binary Operators)
**8.8** 案例研讀: Array類別
**8.9** 物件型別轉換
**8.10** 案例研讀: String類別
**8.11** 多載++與--
**8.12** 案例研讀: Date類別
**8.13** 標準函式庫類別 string 與 vector

# 8.1 簡介

- 使用運算子來處理物件基本運算
  - 運算子多載(operator overloading)
    - 例如:+, -, *, /, %, <<, >>
  - 程式碼較清楚
  - 使用上較方便
    - **object2 = object1.add(object2);**
    - **object2 = object2 + object1;**
    - **object2+= object1;**

# 8.2 運算子多載的基本原理

- 資料型態(Types)
  - 內建資料型態
    - **bool, char, short, int**, long, **float, double**等
      - 可以使用+, -, *, /, <<, >>, ...等運算子
  - 使用者自己定義的物件
    - 不能使用運算子做運算
    - 要使用運算子必須對運算子做多載

# 8.2 運算子多載的基本原理

- 運算子多載
  - 可自行定義使用者物件的運算子運作方式
  - 只能對現有的運算子做多載
  - 不能自己定義新的運算子

# 8.2 運算子多載的基本原理

- 運算子多載的方法
  - 函式命名方式: operator後面跟著運算子
  - 例如:
    - 加法的多載函式名稱為 Operator+
    - 減法的多載函式名稱為 Operator-
    - 乘法的多載函式名稱為 Operator*
    - ...

# 8.3運算子多載的限制

- 運算子多載
  - 不能改變內建資料型態運算子的運作方式
    - **bool, char, short, int**, long**, float, double**等
  - 不能改變運算子的計算優先順序
    - 可使用括號限制運算子的計算順序
  - 不能改變運算方向
    - 由左至右或由右至左
  - 不能改變運算元個數
    - 例如:+,-,*,/有兩個運算元, ++, --有一個運算元
- 每個運算子都必須明確的被多載才能使用
  - 例如:多載 **+**, 並不會自動多載 **+=**

# 8.3運算子多載的限制

- 可被多載的運算子

| Operators that can be overloaded | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | - | * | / | % | ^ | & | \| |
| ~ | ! | = | < | > | += | -= | *= |
| /= | %= | ^= | &= | \|= | << | >> | >>= |
| <<= | == | != | <= | >= | && | \|\| | ++ |
| -- | ->* | , | -> | [] | () | new | delete |
| new[] | delete[] | | | | | | |

- 不可被多載的運算子

| Operators that cannot be overloaded | | | | |
|---|---|---|---|---|
| . | .* | :: | ?: | sizeof |

# 8.4 運算子函式-成員函式與非成員函式

- 運算子函式有兩種實作方式
  - 成員函式(Member functions)
    - 使用this來取得主要運算元
    - 其餘運算元位於參數列上
    - 最左邊的物件型態必須跟運算子的類別相同
    - 例如:Date1+Date2, 則Date1的operator+(Date2)函式被呼叫

# 8.4 運算子函式-成員函式與非成員函式

- 運算子函式有兩種實作方式
  - 非成員函式(Non member functions)
    - 每個運算元皆位於參數列上
    - 最左邊的物件型態可以跟運算元的類別不同
    - 如要存取private或protected資料成員,則必須是friend
    - 例如:Date1+Date2, 則operator+(Date1, Date2)函式被呼叫

# 8.4 運算子函式-成員函式與非成員函式

- 運算子函式呼叫原則
  - 兩個運算元:
    - 呼叫左邊物件的運算子函式
    - 例如:Date1+Date2, 則Date1的operator+(Date2)函式被呼叫
  - 單一運算元:
    - 該物件的運算子被呼叫
    - 例如:Date1++, 則Date1的operator++()函式被呼叫

# 8.4 運算子函式-成員函式與非成員函式

- 交換率(Commutative operators)
  - 例如:
    - “**a + b**”
    - “**b + a**”
  - 要滿足交換率
    - 假設a與b有相同的資料型態
      - 只需寫一個運算子重載函式
    - 假設a與b有不同的資料型態
      - 針對“**a + b**”建立一個運算子重載函式
      - 針對“**b + a**”建立另一個運算子重載函式

# 8.5 多載<<與>>運算子

- << 與 >> 運算子
  - 可供內建資料型態使用
  - 可供使用者自訂類別多載使用
- 多載<<運算子
  - 左邊的運算元必須是型態 **ostream &**
    - 例如: **cout** << Date1
- 多載>>運算子
  - 左邊的運算元必須是型態 **istream &**
    - 例如: **cin** << Date1
- 多載這兩個運算子必須使用 ***非成員函式***

# 8.5 多載<<與>>運算子

- 多載 << 與 >> 運算子範例程式
  - 程式中定義一個類別
    - Class **PhoneNumber**
    - 可供記錄一組電話號碼
  - 多載>>運算子
    - 讓**PhoneNumber**類別可接受輸入格式:(123) 456-7890
  - 多載<<運算子
    - 讓**PhoneNumber**類別可列印輸出格式:(123) 456-7890

Outline

**fig08_03.cpp**
**(1 of 3)**

```
1  // Fig. 8.3: fig08_03.cpp
2  // 多載串流輸入與輸出運算子>>與<<運算子
3  //
4  #include <iostream>
5
6  using std::cout;
7  using std::cin;
8  using std::endl;
9  using std::ostream;
10 using std::istream;
11
12 #include <iomanip>
13
14 using std::setw;
15
16 // 定義PhoneNumber類別
17 class PhoneNumber {
18    friend ostream &operator<<( ostream&, const PhoneNumber & );
19    friend istream &operator>>( istream&, PhoneNumber & );
20
21 private:
22    char areaCode[ 4 ];  // 三個數字區域碼+null
23    char exchange[ 4 ];  // 三個數字交換碼+null
24    char line[ 5 ];      // 四個數字線路碼+null
25
26 }; // end class PhoneNumber
```

Outline

**fig08_03.cpp**
**(2 of 3)**

```
27
28 // 多載運算子<<, 使用非成員函式
29 // 讓物件可以透過<<輸出到cout
30 // cout << somePhoneNumber;
31 ostream &operator<<( ostream &output, const PhoneNumber &num )
32 {
33    output << "(" << num.areaCode << ") "
34           << num.exchange << "-" << num.line;
35
36    return output;     // enables cout << a << b << c;
37
38 } // end function operator<<
39
40 // 多載運算子>>, 使用非成員函式
41 // 讓物件可以透過>>由cin讀取資料
42 // cin >> somePhoneNumber;
43 istream &operator>>( istream &input, PhoneNumber &num )
44 {
45    input.ignore();                    // 跳過一個輸入字元'('
46    input >> setw( 4 ) >> num.areaCode; // 輸入3個區域碼,setw(4)含字串結尾
47    input.ignore( 2 );                 // 跳過二個輸入字元')' ' '
48    input >> setw( 4 ) >> num.exchange; //  輸入3個交換碼
49    input.ignore();                    // 跳過一個輸入字元'-'
50    input >> setw( 5 ) >> num.line;    //  輸入4個線路碼
51
52    return input;      // enables cin >> a >> b >> c;
```

## Outline

**fig08_03.cpp (3 of 3)**

**fig08_03.cpp output (1 of 1)**

```
53
54 } // end function operator>>
55
56 int main()
57 {
58    PhoneNumber phone; // 建立一個屬於PhoneNumber類別的物件
59
60    cout << "Enter phone number in the form (123) 456-7890:\n";
61
62    // cin >> phone 會呼叫 operator>> 非成員函式
63    // operator>>( cin, phone )
64    cin >> phone; // 輸入一個電話號碼
65
66    cout << "The phone number entered was: " ;
67
68    // cout << phone會呼叫 operator<< 非成員函式
69    // operator<<( cout, phone )
70    cout << phone << endl; // 輸出一個電話號碼
71
72    return 0;
73
74 } // end main
```

```
Enter phone number in the form (123) 456-7890:
(800) 555-1212
The phone number entered was: (800) 555-1212
```

# 8.6 多載單運算元運算子

- 多載單運算元運算子(unary operators)
  - 成員函式
    - 沒有輸入參數
    - 需為非靜態(static)成員函式
      - 靜態成員函式只能存取靜態資料
    - 例如:
      - 多載!用來判斷字串是不是空字串
      - **!s 變成 s.operator!()**

```
class String {
   public:
                bool operator!() const;
                ...
};
```

# 8.6 多載單運算元運算子

- 多載單運算元運算子(unary operators)
  - 非成員函式
    - 需要一個輸入參數
    - 參數必須是類別物件或類別物件的參考(reference)
    - 例如:
      - 多載!用來判斷字串是不是空字串
      - **!s 變成 operator!(s)**

```
class String {
        friend bool operator!( const String & )
        ...
}
```

# 8.7多載雙運算元運算子

- 多載雙運算元運算子(Binary Operators)
  - 成員函式
    - 輸入一個參數
    - 需為非靜態(static)成員函式
      - 靜態成員函式只能存取靜態資料
    - 例如:
      - 多載字串相加運算子
      - **class String {**
        **public:**
        **const String &operator+=( const String & );**
        **...**
        **};**
      - **y += z 等於 y.operator+=( z )**

# 8.7多載雙運算元運算子

- 非成員函式
  - 需要兩個輸入參數
  - 例如:
    - 多載字串相加運算子
    - **class String {**
      **friend const String &operator+=(String &, const String & );**
      **...**
      **};**
    - **y += z 等於 operator+=( y, z )**

# 8.8 案例研讀: Array class

- C++的陣列
  - 沒有範圍檢查
  - 不能使用 == 比較
  - 不能用陣列指派命令 =
  - 不能一次輸入/輸出整個陣列

# 8.8 案例研讀: Array class

- 實作一個具以下功能的陣列類別
  - 可檢查範圍
  - 知道陣列大小
  - 可指派陣列(使用=)
  - 可使用[]存取陣列元素
  - 可使用 << 與 >> 輸出/輸入陣列
  - 可使用 == 與 != 做陣列比較

Outline

**array1.h (1 of 2)**

```
1  // Fig. 8.4: array1.h
2  // 定義一個可供存放整數的陣列類別 Array
3  #ifndef ARRAY1_H
4  #define ARRAY1_H
5
6  #include <iostream>
7
8  using std::ostream;
9  using std::istream;
10
11 class Array {
12    friend ostream &operator<<( ostream &, const Array & ); //多載<<
13    friend istream &operator>>( istream &, Array & );       //多載>>
14
15 public:
16    Array( int = 10 );        // 內定建構子
17    Array( const Array & );   // 複製建構子(copy constructor)
18    ~Array();                 // 解構子
19    int getSize() const;      // 傳回陣列大小
20
21    // assignment operator
22    const Array &operator=( const Array & ); //多載=
23
24    // equality operator
25    bool operator==( const Array & ) const;  //多載==
26
```

## Outline

**array1.h (2 of 2)**

```
27    // inequality operator; returns opposite of == operator
28    bool operator!=( const Array &right ) const //多載!=
29    {
30       return ! ( *this == right ); // 叫用Array::operator==
31
32    } // end function operator!=
33
34    // subscript operator for non-const objects returns lvalue
35    int &operator[]( int ); //多載等號左邊的[]
36
37    // subscript operator for const objects returns rvalue
38    const int &operator[]( int ) const; //多載等號右邊的[]
39
40 private:
41    int size; // 陣列大小
42    int *ptr; // 陣列指標
43
44 }; // end class Array
45
46 #endif
```

Outline

**array1.cpp (1 of 7)**

```
1  // Fig 8.5: array1.cpp
2  // Array類別的成員函式定義
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::setw;
12
13 #include <new>        // 使用C++的記憶體配置功能
14
15 #include <cstdlib>    // 使用exit()函式
16
17 #include "array1.h"   // Array類別定義
18
19 // Array類別的內定建構子(內定大小為10)
20 Array::Array( int arraySize )
21 {
22    // 檢查陣列大小
23    size = ( arraySize > 0 ? arraySize : 10 );
24
25    ptr = new int[ size ]; // 配置記憶體供存放陣列資料
26
```

## Outline

**array1.cpp (2 of 7)**

```
27    for ( int i = 0; i < size; i++ )
28       ptr[ i ] = 0;          // 初始化陣列
29
30 } // end Array default constructor
31
32 // Array類別的複製建構子
33 // 輸入必須是一個reference以避免無窮迴圈(因為傳值呼叫會呼叫複製建構子)
34 Array::Array( const Array &arrayToCopy )
35    : size( arrayToCopy.size )
36 {
37    ptr = new int[ size ]; // 配置記憶體空間存放陣列資料
38
39    for ( int i = 0; i < size; i++ )
40       ptr[ i ] = arrayToCopy.ptr[ i ];  // 複製陣列資料
41
42 } // end Array copy constructor
43
44 // Array類別的解構子
45 Array::~Array()
46 {
47    delete [] ptr;  // 釋放陣列的記憶體空間
48
49 } // end destructor
50
```

Outline

**array1.cpp (3 of 7)**

```
51 // 傳回陣列大小
52 int Array::getSize() const
53 {
54    return size;
55
56 } // end function getSize
57
58 // 多載陣算子=(assignment operator)
59 // 傳回型態為const可避免: ( a1 = a2 ) = a3情形下a1的值被改
60 const Array &Array::operator=( const Array &right )
61 {
62    if ( &right != this ) {  // 檢查傳進來的值是否是自己本身
63
64       // 檢查輸入陣列的大小是否與本身大小相同
65       // 如果不同則要重新配置記憶體大小
66       if ( size != right.size ) {
67          delete [] ptr;          // 釋放原來的記憶體空間
68          size = right.size;      // 改變陣列大小
69          ptr = new int[ size ]; // 重新配置記憶體
70
71       } // end inner if
72
73       for ( int i = 0; i < size; i++ )
74          ptr[ i ] = right.ptr[ i ];  // 複製陣列
75
76    } // end outer if
```

Outline

**array1.cpp (4 of 7)**

```
77
78    return *this;   // enables x = y = z, for example
79
80 } // end function operator=
81
82 // 多載運算子==
83 // 檢查兩個陣列內容是否一樣, 一樣則傳回true, 否則傳回false
84 bool Array::operator==( const Array &right ) const
85 {
86    if ( size != right.size )
87       return false;    // 先檢查兩個陣列的大小是否相同
88
89    for ( int i = 0; i < size; i++ )
90
91       if ( ptr[ i ] != right.ptr[ i ] )
92          return false; // 檢查兩個陣列的內容是否相同
93
94    return true;        // 兩個陣列的內容相同
95
96 } // end function operator==
97
```

Outline

**array1.cpp (5 of 7)**

```
98  // 多載等號左邊的運算子[]
99  // 取出註標(subscript)指定位置的陣列元素參考,可供改變該陣列元素內容
100 int &Array::operator[]( int subscript )
101 {
102    // 檢查註標(subscript)是否超過陣列範圍
103    if ( subscript < 0 || subscript >= size ) {
104       cout << "\nError: Subscript " << subscript
105            << " out of range" << endl;
106
107       exit( 1 );  // 註標如果超過陣列範圍則終止程式執行
108
109    } // end if
110
111    return ptr[ subscript ]; // 傳回陣列元素參考
112
113 } // end function operator[]
114
```

Outline

**array1.cpp (6 of 7)**

```
115 // 多載等號右邊的運算子[]
116 // 取出註標(subscript)指定位置的陣列元素參考,不可改變該陣列元素內容
117 const int &Array::operator[]( int subscript ) const
118 {
119    // 檢查註標(subscript)是否超過陣列範圍
120    if ( subscript < 0 || subscript >= size ) {
121       cout << "\nError: Subscript " << subscript
122            << " out of range" << endl;
123
124       exit( 1 );  // 註標如果超過陣列範圍則終止程式執行
125
126    } // end if
127
128    return ptr[ subscript ]; // 傳回陣列元素參考
129
130 } // end function operator[]
131
132 // 多載>>運算子
133 // 輸入整個陣列的值
134 istream &operator>>( istream &input, Array &a )
135 {
136    for ( int i = 0; i < a.size; i++ )
137       input >> a.ptr[ i ];
138
139    return input;   // enables cin >> x >> y;
140
141 } // end function
```

Outline

**array1.cpp (7 of 7)**

```
142
143 // 多載<<運算子
144 ostream &operator<<( ostream &output, const Array &a )
145 {
146    int i;
147
148    // 輸出陣列內容
149    for ( i = 0; i < a.size; i++ ) {
150       output << setw( 12 ) << a.ptr[ i ];
151
152       if ( ( i + 1 ) % 4 == 0 ) // 一列放滿四個數字就換行
153          output << endl;
154
155    } // end for
156
157    if ( i % 4 != 0 )  // 最後一行換行, 只有在最後一行不是4個數字時才換行
158       output << endl;
159
160    return output;   // enables cout << x << y;
161
162 } // end function operator<<
```

Outline

**fig08_06.cpp**
**(1 of 3)**

```
1  // Fig. 8.6: fig08_06.cpp
2  // Array類別測試程式
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include "array1.h"
10
11 int main()
12 {
13    Array integers1( 7 );  // 建立一個可供存放7個元素的陣列
14    Array integers2;       // 建立一個可供存放10個元素的陣列
15
16    // 印出陣列integers1的大小與內容
17    cout << "Size of array integers1 is "
18         << integers1.getSize()
19         << "\nArray after initialization:\n" << integers1;
20
21    // 印出陣列integers2的大小與內容
22    cout << "\nSize of array integers2 is "
23         << integers2.getSize()
24         << "\nArray after initialization:\n" << integers2;
25
```

Outline

fig08_06.cpp
(2 of 3)

```
26    // 輸入17個數字給陣列integers1與陣列integers2
27    cout << "\nInput 17 integers:\n";
28    cin >> integers1 >> integers2;
29
30    cout << "\nAfter input, the arrays contain:\n"
31         << "integers1:\n" << integers1
32         << "integers2:\n" << integers2;
33
34    // 使用陣列運算子!=判斷陣列integers1與陣列integers2是否相同
35    cout << "\nEvaluating: integers1 != integers2\n";
36
37    if ( integers1 != integers2 )
38       cout << "integers1 and integers2 are not equal\n";
39
40    // 使用陣列integers1建立陣列integers3
41    // 列印陣列integers3
42    Array integers3( integers1 );  // 呼叫複製建構子
43
44    cout << "\nSize of array integers3 is "
45         << integers3.getSize()
46         << "\nArray after initialization:\n" << integers3;
47
```

Outline

**fig08_06.cpp**
**(3 of 3)**

```
48    // 使用運算子=設定陣列integers1的值
49    cout << "\nAssigning integers2 to integers1:\n";
50    integers1 = integers2;  // note target is smaller
51
52    cout << "integers1:\n" << integers1
53         << "integers2:\n" << integers2;
54
55    // 使用運算子==判斷陣列integers1是否等於陣列integers2
56    cout << "\nEvaluating: integers1 == integers2\n";
57
58    if ( integers1 == integers2 )
59       cout << "integers1 and integers2 are equal\n";
60
61    // 使用運算子[]讀取陣列內容
62    cout << "\nintegers1[5] is " << integers1[ 5 ];
63
64    // 使用運算子[]設定陣列內容
65    cout << "\n\nAssigning 1000 to integers1[5]\n";
66    integers1[ 5 ] = 1000;
67    cout << "integers1:\n" << integers1;
68
69    // 使用超出範圍的值
70    cout << "\nAttempt to assign 1000 to integers1[15]" << endl;
71    integers1[ 15 ] = 1000;  // ERROR: out of range
72
73    return 0;
74
75 } // end main
```

## Outline

**fig08_06.cpp**
**output (1 of 3)**

```
Size of array integers1 is 7 // 印出陣列integers1的大小與內容
Array after initialization:
          0          0          0          0
          0          0          0

Size of array integers2 is 10 // 印出陣列integers2的大小與內容
Array after initialization:
          0          0          0          0
          0          0          0          0
          0          0

Input 17 integers: // 輸入17個數字給陣列integers1與陣列integers2
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

After input, the arrays contain:
integers1:
          1          2          3          4
          5          6          7
integers2:
          8          9         10         11
         12         13         14         15
```

## Outline

**fig08_06.cpp output (2 of 3)**

```
Evaluating: integers1 != integers2 //判斷陣列integers1與陣列integers2是否相同
integers1 and integers2 are not equal

Size of array integers3 is 7 // 使用陣列integers1建立陣列integers3
Array after initialization:
          1          2          3          4
          5          6          7

Assigning integers2 to integers1: // 使用運算子=設定陣列integers1的值
integers1:
          8          9         10         11
         12         13         14         15
         16         17
integers2:
          8          9         10         11
         12         13         14         15
         16         17

Evaluating: integers1 == integers2 // 判斷陣列integers1是否等於陣列integers2
integers1 and integers2 are equal

integers1[5] is 13 // 使用運算子[]讀取陣列內容
```

## Outline

**fig08_06.cpp**
**output (3 of 3)**

```
Assigning 1000 to integers1[5] // 使用運算子[]設定陣列內容
integers1:
          8          9         10         11
         12       1000         14         15
         16         17

Attempt to assign 1000 to integers1[15] // 使用超出範圍的值

Error: Subscript 15 out of range
```

# 8.9 型別轉換

- 型別轉換
  - 通常用於內定資料型別轉換
    - float to int
    - double to int
    - int to float
    - ...
  - 使用者自訂資料型別(類別) 轉換
    - 要自己定義

# 8.9 型別轉換

- 轉型運算子(Cast operator;conversion operator)
  - 由使用者自訂類別轉到內建資料型態或其它類別
    - class A
    - (char) A, (int) A, (float) A, ...
  - 必須是非靜態(non-static)成員函式
  - 不能使用非成員函式
  - 不用指定傳回值的資料型態

# 8.9 型別轉換

- 例如
  - 將class A轉型成char *
    - 假設s是class A的物件
    - 則**(char *)s**會呼叫**s.operator char*()**
    - Prototype

```
A::operator char *() const;
```

  - 同樣的可將class A轉型成其它內建資料型態或類別

```
A::operator int() const;
A::operator OtherClass() const;
```

# 8.9 型別轉換

- 轉型可減少多載的使用
  - 假設類別**String**可轉型成 **char ***
  - **cout << s; // s is a String**
    - 編譯器會自動將 S 轉型成**char ***
    - 不需要為 S 多載運算子<<
  - 編譯器只能做一個轉型動作

# 8.10 案例研讀: A `String` Class

- 建立類別**`String`**
  - 內含字串建立與字串維護等功能
  - 標準函式庫有提供**`string`**類別
- 轉型建構子(Conversion constructor)
  - 只有一個輸入參數
  - 將其它型態的物件轉換成類別物件
    - 例如:**`String s1("hi");`**
      - 使用char *型態的資料建立**`String`**物件
  - 任何單一輸入參數的建構子都是轉型建構子

Outline

**string1.h (1 of 3)**

```
1  // Fig. 8.7: string1.h
2  // String 類別定義
3  #ifndef STRING1_H
4  #define STRING1_H
5
6  #include <iostream>
7
8  using std::ostream;
9  using std::istream;
10
11 class String {
12    friend ostream &operator<<( ostream &, const String & );
13    friend istream &operator>>( istream &, String & );
14
15 public:
16    String( const char * = "" ); // 轉型建構子,使用char *字串當作輸入
17    String( const String & );    // 複製運算子,使用其它string物件當作輸入
18    ~String();                   // 解構子
19
20    const String &operator=( const String & );  // 多載運算子=,例:S1=S2
21    const String &operator+=( const String & ); // 多載運算子+=,例:S1+=S2
22
23    bool operator!() const;                      // 多載運算子!,用來判斷字串
                                                   // 是不是空的
24    bool operator==( const String & ) const; // 多載運算子==,檢查兩個字串是否相等
25    bool operator<( const String & ) const;  // 多載運算子<, 例:test s1 < s2
26
```

Outline

**string1.h (2 of 3)**

```
27    // 多載運算子!=, 判斷兩字串是否相等,例:test s1 != s2
28    bool operator!=( const String & right ) const
29    {
30       return !( *this == right );
31
32    } // end function operator!=
33
34    // 多載運算子>, 例: s1 > s2
35    bool operator>( const String &right ) const
36    {
37       return right < *this;
38
39    } // end function operator>
40
41    // 多載運算子<=, 例: s1 <= s2
42    bool operator<=( const String &right ) const
43    {
44       return !( right < *this );
45
46    } // end function operator <=
47
48    // 多載運算子>=, 例: s1 >= s2
49    bool operator>=( const String &right ) const
50    {
51       return !( *this < right );
52
53    } // end function operator>=
```

## Outline

**string1.h (3 of 3)**

```
54
55    char &operator[]( int );  // 多載等號左邊運算子[]
56    const char &operator[]( int ) const; // 多載等號右邊運算子[]
57
58    String operator()( int, int ); // 多載運算子(), 用來傳回子字串
59                                   // 參數個數可以是任意個
60    int getLength() const; // 傳回字串長度
61
62 private:
63    int length;  // 用來存放字串長度
64    char *sPtr;  // 指到字串開頭的指標
65
66    void setString( const char * );  // utility function
67                                     // 用來改變字串內容
68 }; // end class String
69
70 #endif
```

Outline

**string1.cpp (1 of 8)**

```
1  // Fig. 8.8: string1.cpp
2  // 類別class的成員函式定義
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <iomanip>
9
10 using std::setw;
11
12 #include <new>          // 使用C++的記憶體配置功能
13
14 #include <cstring>      // 使用strcpy與strcat函式
15 #include <cstdlib>      // 使用exit函式
16
17 #include "string1.h"    // 使用String類別
18
19 // String的轉型建構子:將char *型式的字串轉成String
20 String::String( const char *s )
21    : length( strlen( s ) )
22 {
23    cout << "Conversion constructor: " << s << '\n';
24    setString( s );          // 呼叫設定字串內容函式
25
26 } // end String conversion constructor
```

Outline

**string1.cpp (2 of 8)**

```
27
28 // String的複製建構子
29 String::String( const String &copy )
30    : length( copy.length )
31 {
32    cout << "Copy constructor: " << copy.sPtr << '\n';
33    setString( copy.sPtr ); // 設定字串內容
34
35 } // end String copy constructor
36
37 // 解構子
38 String::~String()
39 {
40    cout << "Destructor: " << sPtr << '\n';
41    delete [] sPtr; // 釋放記憶體
42
43 } // end ~String destructor
44
45 // 多載運算子=, 傳回值使用const String &型態,避免進入無窮迴圈
46 const String &String::operator=( const String &right )
47 {
48    cout << "operator= called\n";
49
50    if ( &right != this ) {          // 檢查是不是自己
51       delete [] sPtr;               // 釋放記憶體
52       length = right.length;        // 改變字串大小
53       setString( right.sPtr );      // 設定字串內容
54    }
```

Outline

**string1.cpp (3 of 8)**

```
55
56    else
57       cout << "Attempted assignment of a String to itself\n";
58
59    return *this; // 啟動多重指派功能, 例:S1=S2=S3;
60
61 } // end function operator=
62
63 //多載運算子+=,
64 //將輸入字串串到原字串後面, 例:S1="ABC", S2="123", S1+=S2, 則S1="ABC123"
65 const String &String::operator+=( const String &right )
66 {
67    size_t newLength = length + right.length;  // 改變長度
68    char *tempPtr = new char[ newLength + 1 ]; // 配置記憶體供存放新字串
69
70    strcpy( tempPtr, sPtr ); //設定tempPtr內容為目前字串物件的內容
71    strcpy( tempPtr + length, right.sPtr ); //將輸入參數的字串內容
72                                            //加到tempPtr後面
73    delete [] sPtr;      // 釋放目前物件的記憶體內容
74    sPtr = tempPtr;      // 令sPtr指標指到tempPtr所指到的記憶體
75    length = newLength; // 改變物件的length內容
76
77    return *this; // 啟動多重指派功能, 例:S1+=S2+=S3;
78
79 } // end function operator+=
80
```

Outline

**string1.cpp (4 of 8)**

```
81 //多載運算子!,用來判斷這個字串是不是空字串
82 bool String::operator!() const
83 {
84    return length == 0;
85
86 } // end function operator!
87
88 //多載運算子==,用來判斷物件的字串是不是跟輸入字串相同
89 bool String::operator==( const String &right ) const
90 {
91    return strcmp( sPtr, right.sPtr ) == 0;
92
93 } // end function operator==
94
95 //多載運算子<,用來比較兩字串的大小
96 bool String::operator<( const String &right ) const
97 {
98    return strcmp( sPtr, right.sPtr ) < 0;
99
100 } // end function operator<
101
```

Outline

**string1.cpp (5 of 8)**

```
102 //多載等號左邊的運算子[]
103 char &String::operator[]( int subscript )
104 {
105    //檢查註標是否超出範圍
106    if ( subscript < 0 || subscript >= length ) {
107       cout << "Error: Subscript " << subscript
108            << " out of range" << endl;
109
110       exit( 1 ); //結束程式
111    }
112
113    return sPtr[ subscript ]; //傳回參考
114
115 } // end function operator[]
116
117 //多載等號右邊的運算子[]
118 const char &String::operator[]( int subscript ) const
119 {
120    //檢查註標是否超出範圍
121    if ( subscript < 0 || subscript >= length ) {
122       cout << "Error: Subscript " << subscript
123            << " out of range" << endl;
124
125       exit( 1 ); //結束程式
126    }
127
128    return sPtr[ subscript ]; //傳回常數參考
129
130 } // end function operator[]
```

Outline

**string1.cpp (6 of 8)**

```
131
132 //傳回由index位置開始長度為length的子字串
133 //例:S1="ABC12345678", 呼叫S1(3,3)會傳回字串"C12"
134 String String::operator()( int index, int subLength )
135 {
136    // 檢查index是否超出字串範圍,或者subLength<0
137    // 如果是的話傳回空字串
138    if ( index < 0 || index >= length || subLength < 0 )
139       return ""; // 傳回空字串
140
141    // 取得子字串長度,如果index+sublength超過字串範圍,則子字串長度需調整
142    int len;
143
144    if ( ( subLength == 0 ) || ( index + subLength > length ) )
145       len = length - index;
146    else
147       len = subLength;
148
149
150    // 配置一塊空間供暫時存放子字串
151    char *tempPtr = new char[ len + 1 ];
152
153    // 將子字串複製到tempPtr並終止字串
154    strncpy( tempPtr, &sPtr[ index ], len );
155    tempPtr[ len ] = '\0';
```

Outline

**string1.cpp (7 of 8)**

```
156
157    //建構一個存放子字串的String物件
158    String tempString( tempPtr );
159    delete [] tempPtr; //釋放記憶體
160
161    return tempString; //傳回存放子字串的String物件
162
163 } // end function operator()
164
165 //傳回字串長度
166 int String::getLength() const
167 {
168    return length;
169
170 } // end function getLenth
171
172 //改變字串內容
173 void String::setString( const char *string2 )
174 {
175    sPtr = new char[ length + 1 ]; //配置記憶體
176    strcpy( sPtr, string2 );       //複製字串
177
178 } // end function setString
```

Outline

**string1.cpp (8 of 8)**

```
179
180 //多載運算元<<
181 ostream &operator<<( ostream &output, const String &s )
182 {
183    output << s.sPtr;
184
185    return output; //允許多重呼叫,例:cout << S1 << S2 << S3
186
187 } // end function operator<<
188
189 //多載運算元>>
190 istream &operator>>( istream &input, String &s )
191 {
192    char temp[ 100 ]; //宣稱一塊記憶體存放輸入字串
193
194    input >> setw( 100 ) >> temp;
195    s = temp; //將char *型態的字串指派給 String物件 s,會使用轉型建構子
196              // const String &String::operator=( const String &right )
197    return input; //允許多重呼叫,例:cin >> S1 >> S2 >> S3
198
199 } // end function operator>>
```

Outline

**fig08_09.cpp**
**(1 of 4)**

```
1  // Fig. 8.9: fig08_09.cpp
2  // String類別測試程式
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include "string1.h" // 使用String類別
9
10 int main()
11 {
12    String s1( "happy" );
13    String s2( " birthday" );
14    String s3;
15
16    // 測試String的==,!=,>,<,>=,<=,<<運算子
17    cout << "s1 is \"" << s1 << "\"; s2 is \"" << s2
18         << "\"; s3 is \"" << s3 << '\"'
19         << "\n\nThe results of comparing s2 and s1:"
20         << "\ns2 == s1 yields "
21         << ( s2 == s1 ? "true" : "false" )
22         << "\ns2 != s1 yields "
23         << ( s2 != s1 ? "true" : "false" )
24         << "\ns2 >  s1 yields "
25         << ( s2 > s1 ? "true" : "false" )
```

Outline

**fig08_09.cpp**
**(2 of 4)**

```
26          << "\ns2 <  s1 yields "
27          << ( s2 < s1 ? "true" : "false" )
28          << "\ns2 >= s1 yields "
29          << ( s2 >= s1 ? "true" : "false" )
30          << "\ns2 <= s1 yields "
31          << ( s2 <= s1 ? "true" : "false" );
32
33       // 測試String的!運算子
34       cout << "\n\nTesting !s3:\n";
35
36       if ( !s3 ) {
37          cout << "s3 is empty; assigning s1 to s3;\n";
38          s3 = s1;
39          cout << "s3 is \"" << s3 << "\"";
40       }
41
42       // 測試String的+=運算子
43       cout << "\n\ns1 += s2 yields s1 = ";
44       s1 += s2;
45       cout << s1;
46
47       // 測試String的轉型建構子
48       cout << "\n\ns1 += \" to you\" yields\n";
49       s1 += " to you"; //測試String的轉型建構子, S1+=(char *)...
50       cout << "s1 = " << s1 << "\n\n";
```

Outline

**fig08_09.cpp**
**(3 of 4)**

```
51
52    // 測試String的()運算子
53    cout << "The substring of s1 starting at\n"
54         << "location 0 for 14 characters, s1(0, 14), is:\n"
55         << s1( 0, 14 ) << "\n\n";
56
57    // 測試String的()運算子,由指定位置到字串結尾
58    cout << "The substring of s1 starting at\n"
59         << "location 15, s1(15, 0), is: "
60         << s1( 15, 0 ) << "\n\n"; // 0表示到字串結尾
61
62    // 測試String的複製建構子
63    String *s4Ptr = new String( s1 );
64    cout << "\n*s4Ptr = " << *s4Ptr << "\n\n";
65
66    // 測試String的=運算子,自己指派給自己
67    cout << "assigning *s4Ptr to *s4Ptr\n";
68    *s4Ptr = *s4Ptr;
69    cout << "*s4Ptr = " << *s4Ptr << '\n';
70
71    // 測試解構子
72    delete s4Ptr;
73
```

Outline

```
74    // 測試String的[]運算子
75    s1[ 0 ] = 'H';
76    s1[ 6 ] = 'B';
77    cout << "\ns1 after s1[0] = 'H' and s1[6] = 'B' is: "
78         << s1 << "\n\n";
79
80    // 測試註標超過範圍的情況
81    cout << "Attempt to assign 'd' to s1[30] yields:" << endl;
82    s1[ 30 ] = 'd';     // ERROR: subscript out of range
83
84    return 0;
85
86 } // end main
```

**fig08_09.cpp**
**(4 of 4)**

Outline

**fig08_09.cpp**
**(1 of 3)**

```
//建構s1, s2, s3三個String物件
Conversion constructor: happy
Conversion constructor:  birthday
Conversion constructor:

// 測試String的==,!=,>,<,>=,<=,<<運算子
s1 is "happy"; s2 is " birthday"; s3 is ""

The results of comparing s2 and s1:
s2 == s1 yields false
s2 != s1 yields true
s2 >  s1 yields false
s2 <  s1 yields true
s2 >= s1 yields false
s2 <= s1 yields true

// 測試String的!運算子
Testing !s3:
s3 is empty; assigning s1 to s3;
operator= called
s3 is "happy"

// 測試String的+=,運算子
s1 += s2 yields s1 = happy birthday
```

Outline

**fig08_09.cpp**
**(2 of 3)**

```
// 測試String的轉型建構子
s1 += " to you" yields
Conversion constructor:  to you
Destructor:  to you
s1 = happy birthday to you

// 測試String的()運算子
Conversion constructor: happy birthday //在()運算子中會建立一個tempString
Copy constructor: happy birthday //()運算子return命令會回傳一個string物件
Destructor: happy birthday //解構tempString

The substring of s1 starting at
location 0 for 14 characters, s1(0, 14), is:
happy birthday

Destructor: happy birthday //解構()運算子回傳的string物件

// 測試String的()運算子,由指定位置到字串結尾
Conversion constructor: to you
Copy constructor: to you
Destructor: to you

The substring of s1 starting at
location 15, s1(15, 0), is: to you

Destructor: to you
```

## Outline

```
// 測試String的複製建構子
Copy constructor: happy birthday to you

*s4Ptr = happy birthday to you

// 測試String的=運算子,自己指派給自己
assigning *s4Ptr to *s4Ptr
operator= called
Attempted assignment of a String to itself
*s4Ptr = happy birthday to you

// 測試解構子
Destructor: happy birthday to you

// 測試String的[]運算子
s1 after s1[0] = 'H' and s1[6] = 'B' is: Happy Birthday to you

// 測試註標超過範圍的情況
Attempt to assign 'd' to s1[30] yields:
Error: Subscript 30 out of range
```

**fig08_09.cpp**
**(3 of 3)**

# 8.11 多載運算子++與--

- 多載前置++與前置--運算子
  - ++object, --object
  - 使用成員函式
    - 函式原型
      - **Date &operator++();**
    - **++d1會呼叫 d1.operator++()函式**
  - 使用非成員函式
    - 函式原型
      - **Friend Date &operator++( Date &);**
    - **++d1會呼叫operator++( d1 )函式**

# 8.11 多載運算子++與--

- 多載後置++與後置--運算子
  - object++或object--
  - 使用成員函式

僅用來區別++或--在後面,參數值沒有特殊意義

    - 函式原型
      - `Date operator++( int );`
    - **d1++會呼叫d1.operator++( 0 )**
  - 使用非成員函式
    - 函式原型
      - `friend Date operator++( Data &, int );`
    - **d1++會呼叫operator++( d1, 0 )**

# 8.11 多載運算子++與--

- 傳回值(Return values)
  - ++object或--object
    - 傳參考 (例:**`Date &`**)
    - 可在等號左邊
  - object ++或object --
    - 傳值
    - 傳回加1或減1前的值
    - 不能在等號左邊

# 8.12 案例研究: A `Date` Class

- **`Date`類別範例**
  - 多載前置++與後置++運算子
    - 改變day, month, 與year
  - 多載+=運算子
  - 檢查是否為閏年
  - 檢查是否為月份的最後一天
  - 檢查是否為年份的最後一天

Outline

**date1.h (1 of 2)**

```
1  // Fig. 8.10: date1.h
2  // Date類別定義
3  #ifndef DATE1_H
4  #define DATE1_H
5  #include <iostream>
6
7  using std::ostream;
8
9  class Date {
10    friend ostream &operator<<( ostream &, const Date & );
11
12 public:
13    Date( int m = 1, int d = 1, int y = 1900 ); // 建構子
14    void setDate( int, int, int ); // 設定日期
15
16    Date &operator++();            // 多載前置++運算子
17    Date operator++( int );        // 多載後置++運算子
18
19    const Date &operator+=( int ); // 多載+=運算子
20
21    bool leapYear( int ) const;    // 檢查是否是閏年
22    bool endOfMonth( int ) const;  // 檢查是否是一個月的最後一天
```

## Outline

**date1.h (2 of 2)**

```
23
24 private:
25    int month;
26    int day;
27    int year;
28
29    static const int days[];        // 存放每個月的天數,整個類別只有一份
30    void helpIncrement();           // 將日期加1並做適當調整
31
32 }; // end class Date
33
34 #endif
```

Outline

**date1.cpp (1 of 5)**

```
1  // Fig. 8.11: date1.cpp
2  // Date類別成員函式定義
3  #include <iostream>
4  #include "date1.h"
5
6
7  // 初始化靜態成員
8  const int Date::days[] =
9     { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
10
11 // 日期建構子
12 Date::Date( int m, int d, int y )
13 {
14    setDate( m, d, y );
15
16 } // end Date constructor
17
18 // 設定月, 日, 年
19 void Date::setDate( int mm, int dd, int yy )
20 {
21    month = ( mm >= 1 && mm <= 12 ) ? mm : 1;
22    year = ( yy >= 1900 && yy <= 2100 ) ? yy : 1900;
23
```

Outline

**date1.cpp (2 of 5)**

```
24    // 檢查是不是閏年
25    if ( month == 2 && leapYear( year ) )
26       day = ( dd >= 1 && dd <= 29 ) ? dd : 1;
27    else
28       day = ( dd >= 1 && dd <= days[ month ] ) ? dd : 1;
29
30 } // end function setDate
31
32 // 多載前置++運算子(例:++d1)
33 Date &Date::operator++()
34 {
35    helpIncrement(); // 將日期加1並做適當調整
36
37    return *this;  // 傳參考
38
39 } // end function operator++
40
41 // 多載後置++運算子(例:d1++)
42 // 整數參數不需要給名稱
43 Date Date::operator++( int )
44 {
45    Date temp = *this; // 複製目前物件狀態
46    helpIncrement(); // 將日期加1並做適當調整
47
48    // 傳回未加1的版本
49    return temp;   // 傳值
50
51 } // end function operator++
```

## Outline

**date1.cpp (3 of 5)**

```
52
53 // 多載+=運算子
54 const Date &Date::operator+=( int additionalDays )
55 {
56    for ( int i = 0; i < additionalDays; i++ )
57       helpIncrement();
58
59    return *this;    // 允許多重呼叫
60
61 } // end function operator+=
62
63 // 假如是閏年傳回 true
64 // 否則傳回 false
65 bool Date::leapYear( int testYear ) const
66 {
67    if ( testYear % 400 == 0 ||
68       ( testYear % 100 != 0 && testYear % 4 == 0 ) )
69       return true;   // 是閏年
70    else
71       return false;  // 不是閏年
72
73 } // end function leapYear
74
```

Outline

**date1.cpp (4 of 5)**

```
75 // 檢查是不是該月的最後一天
76 bool Date::endOfMonth( int testDay ) const
77 {
78    if ( month == 2 && leapYear( year ) )
79       return testDay == 29; // 閏年的最後一天是29日
80    else
81       return testDay == days[ month ];
82
83 } // end function endOfMonth
84
85 // 將日期加1並做適當調整
86 void Date::helpIncrement()
87 {
88    // 日期不是該月的最後一天
89    if ( !endOfMonth( day ) )
90       ++day;
91
92    else
93
94       // 日期是該月的最後一天,而且月份是12月
95       if ( month < 12 ) {
96          ++month;
97          day = 1;
98       }
99
```

## Outline

**date1.cpp (5 of 5)**

```
100       // 日期是該年度的最後一天
101       else {
102          ++year;
103          month = 1;
104          day = 1;
105       }
106
107 } // end function helpIncrement
108
109 // 多載輸運算子
110 ostream &operator<<( ostream &output, const Date &d )
111 {
112    static char *monthName[ 13 ] = { "", "January",
113       "February", "March", "April", "May", "June",
114       "July", "August", "September", "October",
115       "November", "December" };
116
117    output << monthName[ d.month ] << ' '
118           << d.day << ", " << d.year;
119
120    return output;   // 允許多重呼叫
121
122 } // end function operator<<
```

Outline

**fig08_12.cpp**
**(1 of 2)**

```
1   // Fig. 8.12: fig08_12.cpp
2   // Date類別測試程式
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include "date1.h"  // Date類別定義
9
10  int main()
11  {  // 建立日期物件
12     Date d1;  // 內定值是1990年1月1日
13     Date d2( 12, 27, 1992 );
14     Date d3( 0, 99, 8045 );  // 不正確的日期
15     // 印出日期內容,使用<<運算子
16     cout << "d1 is " << d1 << "\nd2 is " << d2
17          << "\nd3 is " << d3;
18     // 使用+=運算子
19     cout << "\n\nd2 += 7 is " << ( d2 += 7 );
20     // 使用前置++運算子
21     d3.setDate( 2, 28, 1992 );
22     cout << "\n\n  d3 is " << d3;
23     cout << "\n++d3 is " << ++d3;
24
25     Date d4( 7, 13, 2002 );
```

Outline

```
26
27    cout << "\n\nTesting the preincrement operator:\n"
28         << "  d4 is " << d4 << '\n';
29    cout << "++d4 is " << ++d4 << '\n';
30    cout << "  d4 is " << d4;
31    // 使用後置++運算子
32    cout << "\n\nTesting the postincrement operator:\n"
33         << "  d4 is " << d4 << '\n';
34    cout << "d4++ is " << d4++ << '\n';
35    cout << "  d4 is " << d4 << endl;
36
37    return 0;
38
39 } // end main
```

**fig08_12.cpp**
**(2 of 2)**

Outline

**fig08_12.cpp**
**output (1 of 1)**

```
// 建立日期物件
d1 is January 1, 1900
d2 is December 27, 1992
d3 is January 1, 1900

// 使用+=運算子
d2 += 7 is January 3, 1993

// 使用後置++運算子
  d3 is February 28, 1992
++d3 is February 29, 1992

Testing the preincrement operator:
  d4 is July 13, 2002
++d4 is July 14, 2002
  d4 is July 14, 2002

// 使用後置++運算子
Testing the postincrement operator:
  d4 is July 14, 2002
d4++ is July 14, 2002
  d4 is July 15, 2002
```

# 8.13 標準函式庫類別string與 vector

- C++內建的類別
  - 可供程式設計者使用
  - **string**
    - 跟我們的**`String`**類別類似
  - **vector**
    - 可動態改變大小的陣列
- 使用**string**與**vector**重做String與 Array範例

# 8.13 標準函式庫類別string與 vector

- 類別**string**
  - Header **<string>**, namespace **std**
  - 有提供轉型建構子
    - **string s1(“hi”);**
  - 有多載 **<<**
    - **cout << s1**
  - 有多載關係運算子
    - **== != >= > <= <**
  - 有多載 = 運算子
  - 有多載 **+=** 運算子

# 8.13 標準函式庫類別string與 vector

- 類別**string**
  - 有多載**[]**運算子
    - 可存取一個字元
    - 沒有範圍檢查
  - 提供子字串函式**substr**
    - **s1.substr(0, 14);**
      - 由第0個位置開始,取得14個字元
    - **S1.substr(15)**
      - 由第15個位置開始到字串結尾
  - 提供**at**函式
    - **s1.at(10)**
      - 傳回第10個字元
    - 有範圍檢查,如果超出範圍會結束程式執行

Outline

**fig08_13.cpp**
**(1 of 4)**

```
1  // Fig. 8.13: fig08_13.cpp
2  // 標準函式庫字串類別測試程式
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <string> // 使用string類別
9
10 using std::string;
11
12 int main()
13 {  // 建立三個string物件
14    string s1( "happy" );
15    string s2( " birthday" );
16    string s3;
17
18    // 測試string的==,!=,>,<,>=,<=,<<運算子
19    cout << "s1 is \"" << s1 << "\"; s2 is \"" << s2
20         << "\"; s3 is \"" << s3 << '\"'
21         << "\n\nThe results of comparing s2 and s1:"
22         << "\ns2 == s1 yields "
23         << ( s2 == s1 ? "true" : "false" )
24         << "\ns2 != s1 yields "
25         << ( s2 != s1 ? "true" : "false" )
```

Outline

**fig08_13.cpp**
**(2 of 4)**

```
26          << "\ns2 >  s1 yields "
27          << ( s2 > s1 ? "true" : "false" )
28          << "\ns2 <  s1 yields "
29          << ( s2 < s1 ? "true" : "false" )
30          << "\ns2 >= s1 yields "
31          << ( s2 >= s1 ? "true" : "false" )
32          << "\ns2 <= s1 yields "
33          << ( s2 <= s1 ? "true" : "false" );
34
35       // 測試string的empty()運算子
36       cout << "\n\nTesting s3.empty():\n";
37
38       if ( s3.empty() ) {
39          cout << "s3 is empty; assigning s1 to s3;\n";
40          s3 = s1;  // assign s1 to s3
41          cout << "s3 is \"" << s3 << "\"";
42       }
43
44       // 測試string的+=運算子,字串串接
45       cout << "\n\ns1 += s2 yields s1 = ";
46       s1 += s2;  // test overloaded concatenation
47       cout << s1;
48
```

Outline

**fig08_13.cpp**
**(3 of 4)**

```
49
50    //測試string的+=運算子,串接C-style字串
51    cout << "\n\ns1 += \" to you\" yields\n";
52    s1 += " to you";
53    cout << "s1 = " << s1 << "\n\n";
54
55    //測試substr函式s1.substr( 0, 14 )
56    cout << "The substring of s1 starting at location 0 for\n"
57         << "14 characters, s1.substr(0, 14), is:\n"
58         << s1.substr( 0, 14 ) << "\n\n";
59
60    //測試substr函式s1.substr( 15 )
61    cout << "The substring of s1 starting at\n"
62         << "location 15, s1.substr(15), is:\n"
63         << s1.substr( 15 ) << '\n';
64
65    //測試複製建構子
66    string *s4Ptr = new string( s1 );
67    cout << "\n*s4Ptr = " << *s4Ptr << "\n\n";
68
69    //測試=運算子,將自己指定給自己
70    cout << "assigning *s4Ptr to *s4Ptr\n";
71    *s4Ptr = *s4Ptr;
72    cout << "*s4Ptr = " << *s4Ptr << '\n';
73
```

Outline

**fig08_13.cpp**
**(4 of 4)**

```
74    // 測試解構子
75    delete s4Ptr;
76
77    // 測試等號左邊的[]運算子
78    s1[ 0 ] = 'H';
79    s1[ 6 ] = 'B';
80    cout << "\ns1 after s1[0] = 'H' and s1[6] = 'B' is: "
81         << s1 << "\n\n";
82
83    // 測試使用at函式註標超過圍的情況
84    cout << "Attempt to assign 'd' to s1.at( 30 ) yields:" << endl;
85    s1.at( 30 ) = 'd';     // ERROR: subscript out of range
86
87    return 0;
88
89 } // end main
```

Outline

**fig08_13.cpp**
**output (1 of 2)**

```
// 建立三個string物件
// 測試string的==,!=,>,<,>=,<=,<<運算子
s1 is "happy"; s2 is " birthday"; s3 is ""

The results of comparing s2 and s1:
s2 == s1 yields false
s2 != s1 yields true
s2 >  s1 yields false
s2 <  s1 yields true
s2 >= s1 yields false
s2 <= s1 yields true
// 測試string的empty()運算子
Testing s3.empty():
s3 is empty; assigning s1 to s3;
s3 is "happy"
// 測試string的+=運算子,字串串接
s1 += s2 yields s1 = happy birthday
//測試string的+=運算子,串接C-style字串
s1 += " to you" yields
s1 = happy birthday to you
//測試substr函式s1.substr( 0, 14 )
The substring of s1 starting at location 0 for
14 characters, s1.substr(0, 14), is:
happy birthday
```

## Outline

**fig08_13.cpp**
**output (2 of 2)**

```
//測試substr函式s1.substr( 15 )
The substring of s1 starting at
location 15, s1.substr(15), is:
to you
//測試複製建構子(string *s4Ptr = new string( s1 ))
*s4Ptr = happy birthday to you
//測試=運算子,將自己指定給自己
assigning *s4Ptr to *s4Ptr
*s4Ptr = happy birthday to you
// 測試解構子
// 測試等號左邊的[]運算子
s1 after s1[0] = 'H' and s1[6] = 'B' is: Happy Birthday to you
// 測試使用at函式註標超過圍的情況
Attempt to assign 'd' to s1.at( 30 ) yields:

abnormal program termination //程式不正常結束
```

# 8.13 標準函式庫類別string與 vector

- 類別**vector**
  - Header **<vector>**, namespace **std**
  - 可供建立各種資料型態的陣列
    - **vector< int > myArray(10)**
  - 提供**size**(**myArray.size()**)
  - 多載運算子 **[]**
    - 取得指定位置元素, **myArray[3]**
  - 多載運算子 **!=**, **==**, 與 **=**

Outline

**fig08_14.cpp**
**(1 of 5)**

```
1  // Fig. 8.14: fig08_14.cpp
2  // 使用標準函式庫類別vector
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  #include <iomanip>
10
11 using std::setw;
12
13 #include <vector> // 使用標準函式庫類別vector
14
15 using std::vector;
16
17 void outputVector( const vector< int > & );
18 void inputVector( vector< int > & );
19
20 int main()
21 {
22    vector< int > integers1( 7 );   // 建構內含7個int元素的陣列
23    vector< int > integers2( 10 );  // 建構內含10個int元素的陣列
24
```

Outline

**fig08_14.cpp**
**(2 of 5)**

```
25    // 印出 integers1 的大小與內容
26    cout << "Size of vector integers1 is "
27         << integers1.size()
28         << "\nvector after initialization:\n";
29    outputVector( integers1 );
30
31    // 印出 integers2 的大小與內容
32    cout << "\nSize of vector integers2 is "
33         << integers2.size()
34         << "\nvector after initialization:\n";
35    outputVector( integers2 );
36
37    // 輸入與印出 integers1 與 integers2
38    cout << "\nInput 17 integers:\n";
39    inputVector( integers1 );
40    inputVector( integers2 );
41
42    cout << "\nAfter input, the vectors contain:\n"
43         << "integers1:\n";
44    outputVector( integers1 );
45    cout << "integers2:\n";
46    outputVector( integers2 );
47
48    // 使用運算子不等於 (!=)
49    cout << "\nEvaluating: integers1 != integers2\n";
50
```

Outline

**fig08_14.cpp**
**(3 of 5)**

```
51    if ( integers1 != integers2 )
52       cout << "integers1 and integers2 are not equal\n";
53
54    // 使用integers1 建構向量 integers3
55    // 印出integers3的大小與內容
56    vector< int > integers3( integers1 );  // 複製建構子
57
58    cout << "\nSize of vector integers3 is "
59         << integers3.size()
60         << "\nvector after initialization:\n";
61    outputVector( integers3 );
62
63
64    // 使用運算子(=)
65    cout << "\nAssigning integers2 to integers1:\n";
66    integers1 = integers2;
67
68    cout << "integers1:\n";
69    outputVector( integers1 );
70    cout << "integers2:\n";
71    outputVector( integers1 );
72
```

Outline

**fig08_14.cpp**
**(4 of 5)**

```
73    // 使用運算子(==)
74    cout << "\nEvaluating: integers1 == integers2\n";
75
76    if ( integers1 == integers2 )
77       cout << "integers1 and integers2 are equal\n";
78
79    // 使用等號右邊的運算子[]
80    cout << "\nintegers1[5] is " << integers1[ 5 ];
81
82    // 使用等號左邊的運算子[]
83    cout << "\n\nAssigning 1000 to integers1[5]\n";
84    integers1[ 5 ] = 1000;
85    cout << "integers1:\n";
86    outputVector( integers1 );
87
88    // 使用超出範圍的註標
89    cout << "\nAttempt to assign 1000 to integers1.at( 15 )"
90         << endl;
91    integers1.at( 15 ) = 1000;  // ERROR: out of range
92
93    return 0;
94
95 } // end main
96
```

Outline

**fig08_14.cpp**
**(5 of 5)**

```
97  // output vector contents
98  void outputVector( const vector< int > &array )
99  {
100    for ( int i = 0; i < array.size(); i++ ) {
101       cout << setw( 12 ) << array[ i ];
102
103       if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
104          cout << endl;
105
106    } // end for
107
108    if ( i % 4 != 0 )
109       cout << endl;
110
111 } // end function outputVector
112
113 // input vector contents
114 void inputVector( vector< int > &array )
115 {
116    for ( int i = 0; i < array.size(); i++ )
117       cin >> array[ i ];
118
119 } // end function inputVector
```

Outline

**fig08_14.cpp**
**output (1 of 2)**

```
// 建構內含7個int元素的陣列integers1
// 建構內含10個int元素的陣列integers2

// 印出 integers1 的大小與內容
Size of vector integers1 is 7
vector after initialization:
           0           0           0           0
           0           0           0

// 印出 integers2 的大小與內容
Size of vector integers2 is 10
vector after initialization:
           0           0           0           0
           0           0           0           0
           0           0

// 輸入與印出 integers1 與 integers2
Input 17 integers:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

After input, the vectors contain:
integers1:
           1           2           3           4
           5           6           7
integers2:
           8           9          10          11
          12          13          14          15
          16          17
```

```
// 使用運算子不等於 (!=)
Evaluating: integers1 != integers2
integers1 and integers2 are not equal

// 使用integers1 建構向量 integers3
// 印出integers3的大小與內容
Size of vector integers3 is 7
vector after initialization:
          1           2           3           4
          5           6           7

// 使用運算子(=)
Assigning integers2 to integers1:
integers1:
          8           9          10          11
         12          13          14          15
         16          17

integers2:
          8           9          10          11
         12          13          14          15
         16          17

// 使用運算子(==)
Evaluating: integers1 == integers2
integers1 and integers2 are equal
```

## Outline

**fig08_14.cpp**
**output (2 of 2)**

```
// 使用等號右邊的運算子[]
integers1[5] is 13

// 使用等號左邊的運算子[]
Assigning 1000 to integers1[5]
integers1:
           8           9          10          11
          12        1000          14          15
          16          17

// 使用超出範圍的註標
Attempt to assign 1000 to integers1.at( 15 )

abnormal program termination
```

## Outline

**fig08_14.cpp**
**output (2 of 2)**

# Card 類別

Outline

```
#ifndef _CARD_H_
#define _CARD_H_

#define NUM_OF_CARD  52
#define NUM_OF_COLOR 4
#define NUM_OF_POINT 13

//定義撲克牌(Card)類別
class Card{

public:
          Card(int color, int point);//建構子
          void print(void) const;    //列印卡片內容
          int getColor(void) const;  //傳回花色
          int getPoint(void) const;  //傳回點數

private:
          const int color ; //花色
          const int point ; //點數
};

#endif
```

Outline

# Dealer 類別

```
#ifndef _DEALER_H_
#define _DEALER_H_

#include "card.h"
#include "Player.h"

// 定義發牌員(Dealer)類別
class Dealer{

public:
    Dealer(void);                 //建構子
    void pushCard(Card &c);       //拿一張牌給發牌員
    void shuffle(void);           //洗牌
    void deal(int n, Player &p); //發n張牌給玩家p
    void print() const;           //印出發牌員手上的牌

private:
    Card &popCard(void);      //從發牌員手上拿取一張牌
    Card *card[NUM_OF_CARD]; //用來存放撲克牌
    int  numCard;             //發牌員手上的撲克牌數

};

#endif
```

# Player 類別

Outline

```
#ifndef _PLAYER_H_
#define _PLAYER_H_

#include "card.h"

// 定義玩家(Player)類別
class Player{

public:
          Player(int m=1000);           //建構子
          void pushCard(Card &);        //拿一張牌給玩家
          Card &popCard();              //從玩家手上拿取一張牌
          void print(void) const;       //印出玩家手上的牌
          void printMoney(void) const; //印出玩家手上的錢
          void sort(void);              //將玩家手上的牌排序
          void setBet(int);             //設定賭注金額
          int  getBet(void) const;      //取得賭注金額
          int  getMoney(void) const;    //取得玩家手上的錢數
          int  showHand(void);          //顯示並傳回此回合結果,計算(賭注x倍率)
                                        //將(賭注x倍率)加到玩家手上的錢數
private:
          Card *card[NUM_OF_CARD];      //存放撲克牌
          int numCard;                  //存放玩家手上的撲克牌數
          int money;                    //存放玩家錢數
          int bet;                      //存放賭注
};

#endif
```

```
int PRIZE_TABLE[9]={//倍率表
                    0,      //沒中獎
                 1000,      //同花順
                  500,      //鐵支
                  300,      //同花
                  200,      //葫蘆
                  100,      //順子
                   20,      //三條
                    5,      //二對
                    1       //一對
};
```