自定輸出格式處理器

使用者也可以自行設定輸入 / 輸出格式處理器。舉例來說，由於字元 (char) 僅佔用一個位元組的記憶空間，因此有時我們也可將字元當成一個位元組的整數來使用。但很不幸的，字元變數的輸出除非轉型否則一律是以字元方式呈現，例如：

```
char  no = 67 ;
cout << no << endl ;                        // 輸出 C 字元
cout << static_cast<int>(no) << endl ;   // 轉型為整數後輸出 67
```

雖然在程式中可以利用轉型的方式間接地輸出字元所對應的整數，但作法有點過於強烈，一種較緩和的方式是將之隱藏於輸出格式處理器內，例如：

```
cout << chint << no << endl ;    // 輸出整數 67
```

這裡的 chint 為須要定義的輸出格式處理器，其作用就是要將緊接於其後的字元變數以整數的方式輸出。為達到此目的，首先定義一個簡單結構：

```
struct  Chint {
    ostream  *optr ;   // 指標 optr 指向 ostream 輸出物件
} ;

Chint  chint ;         // chint 輸出格式處理器為 Chint 結構的物件
```

此 Chint 結構內僅有一個指標 optr 用來指向 ostream 輸出物件，而 chint 輸出格式處理器則只是此結構的一個物件。仔細觀察 cout << chint << no << endl 式子的前兩個輸出運算子的型式，我們可以分別定義以下兩種不同型式的輸出運算子：

```
Chint&  operator<< ( ostream& out , Chint& foo ) {
    foo.optr = &out ;
    return  foo ;
}

ostream&  operator<< ( Chint& foo , char c ) {
    return  *(foo.optr) << static_cast<int>(c) ;
}
```

以上設計看似複雜，但也仍是運算子覆載的應用而矣。前一個輸出運算子讓 Chint 格式處理器的指標指向 ostream 物件，而後者只是將轉型式子包裹於其內。

六角形數字盤

在這個簡單的程式中，我們要利用設定背景字元與輸出的寬度來列印六角形的數字盤，並且在不同數字間以空白字元加以分開。在程式中，我們特別設定背景字元與輸出的字串字元相同，造成在輸出寬度內的所有格子都是列印同樣的字元，如此可以巧