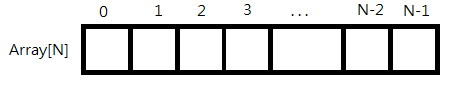
***Arrays***

所謂array是指一系列元素，有著固定大小。因此你無法藉由增加或移除元素而改變其大小，它只允許你替換元素值。

如果你需要一個有固定元素量的序列，class array<>將帶來最佳效能，因為記憶體被分配於stack中，絕不會被重分配(reallocation)。



***Vectors***

vectors模塑出一個dynamic array。它本身是「將元素置於dynamic array中加以管理」的一個抽象概念。不過，C++ Standard 並未要求必須以dynamic array實作vector，僅僅只是規定其相應條件和操作複雜度。

如果你在末端附加或刪除元素，vector的效率相當好。但如果你在前端或中段安插或刪除元素，效率就不怎麼樣了，因為作用點之後的每一個元素都必須移到另一個位置，而每一次移動都得呼叫assignment運算子。

vector優異效能的秘訣之一，就是分配出「較其容納的元素」更多的記憶體。為了能夠高效運用vectors，應該瞭解大小和容量之間的關係。

vectors提供用以操作大小的函式有size(), empty(), max\_size()。另一個與大小有關的函式是capacity()，返回vector實際能夠容納的元素量。如果超越這個量，vector就有必要重新分配內部記憶體。

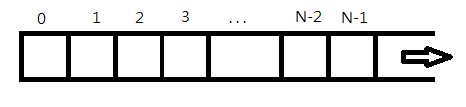
vector的容量之所以重要，有2個原因：

1. 一旦記憶體重新分配，vector元素相關的所有reference、pointers、iterators，都會失效。
2. 記憶體重新分配很耗時間。

所以如果你的程式管理了「與vector元素相關」的reference、pointers、iterators，或如果程式的執行速度對你而言至關重要，那就必須考慮容量問題。

你可以使用reserve()保留適當容量，避免重新分配記憶體。如此一來，只要保留的容量尚有餘裕，就不必擔心reference失效。

vectors的容量概念與strings類似，但有一個大不同：vector不能使用reserve()縮減容量，這一點和strings不同。



***Deques***

容器deque和vector非常相似。它也採用dynamic array來管理元素，提供隨機存取，並有著和vector幾乎一模一樣的介面。不同的是deque的dynamic array頭尾都開放，因此能在頭尾兩端進行快速安插和刪除。

為了提供這種能力，deque通常實作為一組獨立區塊(a bunch of individual blocks)，第一區塊朝某方向發展，最末區塊朝另一方向擴展。

deques與vectors相比，功能上的差異如下：

* 兩端都能快速安插元素和移除元素(vector只能在尾端)。這些操作可以在攤提的時間(amortized constant time)內完成。
* 存取元素時deque內部結構會多一個間接過程，所以元素的存取和迭代器的動作會稍稍慢一些。
* 迭代器需要在不同區塊間跳轉，所以必須是個smart pointer，不能是尋常pointers。
* 在記憶體區塊大小有限制的系統中(例如PC系統)，deque可以內含更多元素，因為它使用不只一塊記憶體。因此deque的max\_size()可能更大。
* deque不支援對容量和記憶體重新分配時機的控制。特別要注意的是，除了頭尾兩端在任何地點安插或刪除元素都將導致指向deque的任何pointers、references、iterators失效。不過，deque的記憶體重分配優於vector，因為其內部結構顯示，deque不必在記憶體重新分配時複製所有元素。
* deque會釋放不再使用的記憶體區塊。deque的記憶體大小是可縮減的，但要不要這麼做，以及如何做由實作件決定。

deques的以下特性跟vectors差不多：

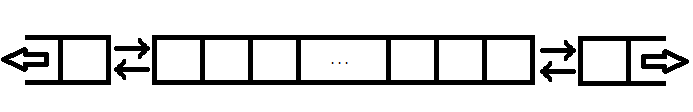
* 在中段安插、移除元素的速度相對較慢，因為所有元素都需移動以騰出或填補空間。
* 迭代器屬於random access iterator(隨機存取迭代器)。

總之，以下情形最好採用deque：

* 你需要在兩端安插和移除元素。
* 無須指涉(refer to)容器內的元素
* 要求「不再使用的元素必須釋放」(不過C++Standard對此無任何保證)。

deques的各項操作只有以下兩點和vectors不同：

* deques不提供容量操作(capacity()和reserve())。
* deques直接提供函式完成頭部元素的安插和刪除(push\_front()和pop\_front())。



***Lists***

List(我說的是容器類別list<>的一份實體)使用一個doubly linked list(雙向串列)管理元素。C++標準庫並未明定實作方式，只是遵守list的名稱、限制和規格。

list的元素可以是任何型別T。預設的記憶體模型是C++標準庫提供的allocator。

Lists的內部構造完全迥異於array，vector或deque。List物件自身提供了兩個pointers，或稱anchors(錨點)，用來指涉第一個和最末一個元素。每一個元素都有pointer指向前一個和下一個元素(或是指回anchor)。如果想要安插新元素，只需操縱對應的pointer即可。

因此，List在幾個主要方面與arrays, vector 或deque不同：

* List不支援隨機存取。如果你要存取第5個元素，就得順著串鏈逐一爬過前4個元素。所以，在list中隨機尋訪任意元素是很緩慢的行為。然而你可以從兩端開始航行整個list，所以存取第一個或最末一個元素的速度很快。
* 任何位置上(不只兩端)執行元素的安插和移除都非常快，始終都是常數時間內完成，因為無須移動任何其他元素。實際上內部指示進行一些pointers操作而已。
* 安插和刪除動作並不會造成指向其他元素的各個pointers、references、iterators失效。
* List對於異常的處理方式是：要嘛操作成功，要嘛甚麼都不發生。你絕不會陷入「只成功一半」這種進退維谷的尷尬境地。

Lists所提供的成員函式反映出它和arrays, vector 或deque的不同：

* Lists提供front(), push\_front(), pop\_front(), back(), push\_back(), pop\_back()等操作函式。
* 由於不支援隨機存取，lists既不提供subscript(下標)運算子，也不提供at()。
* Lists並未提供容量、空間重新分配等操作函式，因為全無必要。每個元素有自己的記憶體，在元素被刪除前一直有效。
* Lists提供不少特殊成員函式，專門用於移動和移除元素。較之同名的STL通用演算法，這些函式執行起來更快速，因為它們無須拷貝或搬移元素，只需調整若干pointers即可。

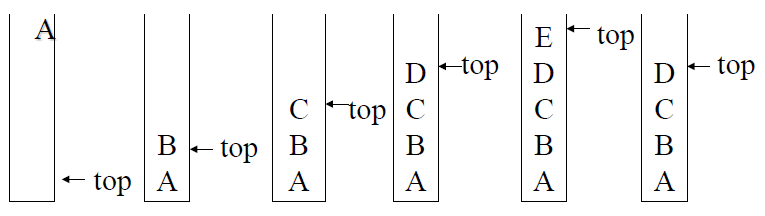


***Stacks***

class stack<>實作出一個stack(也稱為LIFO，後進先出)。你可以使用push()將任意數量的元素置入stack中，也可以使用pop()將元素依其插入的相反次序從容器中移除(此即所謂「後進先出」，LIFO)。

在表頭檔<stack>，class stack的定義中第一個template參數代表元素型別。帶有預設值的第二個template參數用來定義stack內部用來存放元素的實際容器，預設為deque。之所以選擇deque而非vector，是因為deque移除元素時會釋放記憶體，並且不必再重分配(reallocation)實複製全部元素。

Stack的實作件中只是很單純地把各項操作轉化為內部容器的對應呼叫。你可以使用任何Sequence容器支援stack，只要它們提供以下成員函式：back(), push\_back()和pop\_back。例如你可以使用vector或list來容納元素。



***Queues***

class queue<>實作出一個queue(也稱為FIFO，後進先出)。你可以使用push()將任意數量的元素置入queue中，也可以使用pop()將元素依其插入次序從容器中移除(此即所謂「先進先出」，FIFO)。

在表頭檔<queue>，class queue的定義中第一個template參數代表元素型別。帶有預設值的第二個template參數用來定義deque內部用來存放元素的實際容器，預設為deque。

實際上queue只是很單純地把各項操作轉化為內部容器的對應呼叫。你可以使用任何Sequence容器支援queue，只要它們提供以下成員函式：front(), back(), push\_back()和pop\_front。例如你可以使用list來容納元素。

