



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

Vectors (Vectores)

Agustín J. González

ELO329

¿Qué es un vector?

- De la biblioteca de plantillas estándares de C++ (standard template library) (STL):
 - Un vector es una secuencia que soporta accesos aleatorios a elementos, posee tiempo constante en inserción y eliminación de elementos de los extremos, y tiempo lineal en inserciones y eliminaciones de elementos en el medio. El número de elementos en un vector puede variar dinámicamente; administración de memoria es automática.
- El vector es la clase contenedora más simple de la STL y en muchos casos la más eficiente.

Table 8.1 Operations for the `vector` data type

Constructors		
<code>vector<T> v;</code>	Default constructor	$O(1)$
<code>vector<T> v (int);</code>	Initialized with explicit size	$O(n)$
<code>vector<T> v (int, T);</code>	Size and initial value	$O(n)$
<code>vector<T> v (aVector);</code>	Copy constructor	$O(n)$
Element Access		
<code>v[i]</code>	Subscript access, can be assignment target	$O(1)$
<code>v.front ()</code>	First value in collection	$O(1)$
<code>v.back ()</code>	Last value in collection	$O(1)$
Insertion		
<code>v.push_back (T)</code>	Push element on to back of vector	$O(1)$ ^a
<code>v.insert(iterator, T)</code>	Insert new element after iterator	$O(n)$
<code>v.swap(vector<T>)</code>	Swap values with another vector	$O(n)$
Removal		
<code>v.pop_back ()</code>	Pop element from back of vector	$O(1)$
<code>v.erase(iterator)</code>	Remove single element	$O(n)$
<code>v.erase(iterator, iterator)</code>	Remove range of values	$O(n)$
Size		
<code>v.capacity ()</code>	Maximum elements buffer can hold	$O(1)$
<code>v.size ()</code>	Number of elements currently held	$O(1)$
<code>v.resize (unsigned, T)</code>	Change to size, padding with value	$O(n)$
<code>v.reserve (unsigned)</code>	Set physical buffer size	$O(n)$
<code>v.empty ()</code>	True if vector is empty	$O(1)$
Iterators		
<code>vector<T>::iterator itr</code>	Declare a new iterator	$O(1)$
<code>v.begin ()</code>	Starting iterator	$O(1)$
<code>v.end ()</code>	Ending iterator	$O(1)$

Interfaz de Plantilla Vector (Ver web)

Algoritmos Estándares útiles en Vectores

- Encontrar un valor único
- Contar el número de coincidencias
- Recolectar todos los valores coincidentes
- Remover un elemento
- Insertar un elemento

Table 8.2 Generic algorithms useful with vectors

`fill (iterator start, iterator stop, value)`

Fill vector with a given initial value

`copy (iterator start, iterator stop, iterator destination)`

Copy one sequence into another

`max_element(iterator start, iterator stop)`

Find largest value in collection

`min_element(iterator start, iterator stop)`

Find smallest value in collection

`reverse (iterator start, iterator stop)`

Reverse elements in the collection

`count (iterator start, iterator stop, target value, counter)`

Count elements that match target value, incrementing counter

`count_if (iterator start, iterator stop, unary fun, counter)`

Count elements that satisfy function, incrementing counter

`transform (iterator start, iterator stop, iterator destination, unary)`

Transform elements using unary function from source, placing into destination

`find (iterator start, iterator stop, value)`

Find value in collection, returning iterator for location

`find_if (iterator start, iterator stop, unary function)`

Find value for which function is true, returning iterator for location

`replace (iterator start, iterator stop, target value, replacement value)`

Replace target element with replacement value

`replace_if (iterator start, iterator stop, unary fun, replacement value)`

Replace elements for which fun is true with replacement value

`sort (iterator start, iterator stop)`

Places elements into ascending order

`for_each (iterator start, iterator stop, function)`

Execute function on each element of vector

`iter_swap (iterator, iterator)`

Swap the values specified by two iterators

■ Algoritmos
Genéricos ya
implementados
en la STL

■ Pueden ser
usados con
vectores y
otros
contenedores
de la STL

■ (Ver WEB)

Algoritmos Genéricos ya implementados en la STL (continuación)

Table 8.3 Generic Algorithms useful with Sorted Vectors

`merge(iterator s1, iterator e1, iterator s2, iterator e2, iterator dest)`

Merge two sorted collections into a third

`inplace_merge(iterator start, iterator center, iterator stop)`

Merge two adjacent sorted sequences into one

`binary_search (iterator start, iterator stop, value)`

Search for element within collection, returns a boolean

`lower_bound (iterator start, iterator stop, value)`

Find first location larger than or equal to value, returns an iterator

`upper_bound (iterator start, iterator stop, value)`

Find first element strictly larger than value, returns an iterator

Ordenar un Vector

- Suponer que el vector contiene ítems cuyo tipo/clases es predefinido en C++
- La función `sort()` pertenece a la biblioteca `<algorithm>`
- `begin()` apunta al primer elemento, y `end()` apunta a la posición siguiente al último elemento

```
#include <algorithm>
vector<int> items;
// otras operaciones
sort( items.begin(), items.end());
```

Iteradores (Iterators)

- Un iterador es un puntero a un elemento de una clase contenedora. Puede ser visto como la generalización del índice de un arreglo. El iterador puede ser movido hacia delante o hacia atrás a través de los elementos del contenedor.
- La clase `vector` es un ejemplo de contenedor en la STL.
- Desreferenciamos un iterador para acceder al elemento que éste apunta. (* = operador de desreferencia o “valor apuntado por”)
- `vector<int> items;`
- `vector<int>::iterator I;`
- `I = items.begin(); // first number`
- `cout << *I << endl; // display the first number`

Ordenamiento Usando Iteradores

- Podemos pasar iteradores a la función `sort()`

```
#include <algorithm> // ver funciones de algorithm en STL
```

```
vector<int> items;
```

```
vector<int>::iterator I1;
```

```
vector<int>::iterator I2;
```

```
I1 = items.begin();
```

```
I2 = items.end();
```

```
sort( I1, I2 );
```

Sort sobre Tipos definidos por el usuario

- Ordenar un vector que contenga elementos de nuestra clase es levemente más avanzado
- Debemos sobrecargar el operador < en nuestra clase

```
vector<Student> elo329;
```

```
sort( elo329.begin(), elo329.end());
```

Sobrecarga del Operador <

```
class Student {  
public:  
    bool operator <(const Student & S2) {  
        return m_sID < S2.m_sID;  
    }  
  
private:  
    string m_sID;  
    string m_sLastName;  
};
```

Operaciones comunes con vectores

```
vector<int> items;
```

```
// Reverse the order
```

```
reverse( items.begin(), items.end());
```

```
// Randomly shuffle the order
```

```
random_shuffle( items.begin(), items.end());
```

```
// Accumulate the sum
```

```
#include <numeric>
```

```
int sum = accumulate( items.begin(), items.end(), 0 );
```

Encontrar/Remove el valor más pequeño

```
vector<int> items;
```

```
vector<int>::iterator I;
```

```
// find lowest value
```

```
I = min_element(items.begin(),items.end());
```

```
// erase item pointed to by iterator I
```

```
items.erase( I );
```