



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA



Vectors (Vectores)

Agustín J. González
ELO329



Contenidos

- Qué es un vector?
- Declaración de Objetos Vectores
- Inserción y eliminación de ítemes
- Uso de sub-índices
- Obtención y modificación del tamaño
- Pre-asignación de espacio
- Paso de vectores a función
- Retorno de instancias de vector



Qué es un vector?

- De la biblioteca de plantillas estándares de C++ 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 al comienzo o 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.

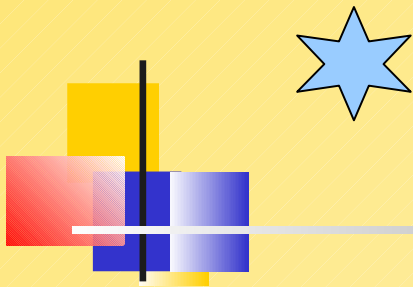


Table 8.1 Operations for the `vector` data type

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	$O(1)$

^a `push_back` can be $O(n)$ if reallocation of the internal buffer is necessary, otherwise it is $O(1)$.

Interfaz de Plantilla Vector



Declaración de Objetos Vector



- Podemos declarar vectores de cualquier tipo
 - El vector puede estar vacío o puede tener un tamaño.

```
#include <vector>
```

```
vector<double> scores(20);
```

```
vector<string> names;
```

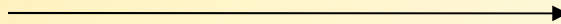
```
vector<bool> busyFlags(5);
```

```
vector<Student> classRoll(50);
```



Inserción y Eliminación de Ítem

- `push_back(item)` inserta un ítem
- `pop_back()` elimina un ítem, pero no lo retorna





Inserción y Eliminación de ítems

```
vector<double> temps;
```

```
temps.push_back( 88.5 );
```

```
temps.push_back( 87.2 );
```

```
temps.push_back( 82.1 );
```

```
// now the vector contains elements in
```

```
// positions [0], [1], and [2].
```

```
// remove the last element:
```

```
temps.pop_back();
```



Uso de Sub-índices

- Para cualquier sub-índice n , lo siguiente debe ser verdadero:

`0 <= n < size()`

- La case de vectores en C++ no atrapan este error !!

```
vector<int> scores;
```

```
scores[0] = 25;           // error
```

```
scores.push_back( 15 );
```

```
scores[0] = 25;         // ok now
```




Constructor

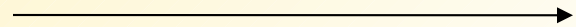
- Un constructor de un vector puede tomar un parámetro entero que define su tamaño
- Cada elemento puede ser opcionalmente inicializado por el constructor

```
vector<string> names(10);  
  
vector<int> scores(10, 0);  
// all 10 elements contain zero  
  
vector<int> backup( scores );  
// make a copy of a vector
```



Obtención y Cambio del tamaño

- `size()` retorna el número de elementos en el vector
- `empty()` retorna verdadero si el tamaño es cero
- `push_back()` aumenta el tamaño en 1
- `pop_back()` reduce el tamaño en 1
- `resize()` cambia el tamaño





Obtención y Cambio del tamaño

```
vector<string> names(10);
```

```
cout << names.size(); // 10
```

```
names.push_back("Sam");
```

```
cout << names.size(); // 11
```

```
names.resize(15); // size = 15
```

```
names.pop_back(); // size = 14
```



Expansión Automática

- `push_back()` causa que el vector aumente su espacio asignado si es necesario
- Una copia del vector es hecha, lo cual causa overhead (tiempo de ejecución no usado eficientemente)

Reserva de Espacio

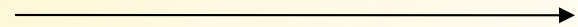
- `reserve(n)` reserva un espacio para expansión sin afectar el valor retornado por `size()`
- Usamos `reserve()` para hacer `push_back()` más eficiente



Vectores en Clases

Un vector debería ser encapsulado en una clase para proveer adecuado chequeo de errores

```
class Scores {  
public:  
    double At(int index) const;  
    // return the score at index  
  
private:  
    vector<double> m_vScores;  
};
```





Vectores en Clases

La función `At()` puede efectuar chequeo de rango:

```
double At(int index) const
{
    if(index >= 0 && index < m_vScores.size())
        return m_vScores[index];
    else
        return 0;
}
```



Vectores en Clases

Un valor de tamaño no puede ser pasado a un constructor de un vector en una clase:

```
class MyClass {  
public:  
  
private:  
    vector<int> myVec(20); // error!  
    vector<int> myVec;    // ok  
};
```



Vectores en Clases

- En su lugar, el espacio para el vector puede ser reservado en la implementación del constructor usando el iniciador de miembros:

```
MyClass::MyClass(int theSize)
    : myVec(theSize)
{

}
```




Vectores en Clases

- O, el espacio para el vector puede ser reservado en el cuerpo del constructor:

```
MyClass::MyClass(int theSize)
{
    myVec.reserve(theSize);
    // size() still returns 0
}
```



Vectores en Clases

- O, el tamaño puede ser explícitamente definido, lo cual causa la inserción de objetos vacíos:

```
MyClass::MyClass(int theSize)
{
    myVec.resize(theSize);


    // size() returns value of theSize
}
```



Paso de Vectores a Funciones

- Siempre pasemos un vector por referencia
- Usamos const si el vector no será modificado

```
double calcAverage(const vector<double> & scores )  
{  
    double avg = 0;  
  
    //...  
  
    return avg;  
}
```





Llenado de un Vector

- Ejemplo: Llenado de un vector con enteros aleatorios entre 0 y 99:

```
void fillRandom( vector<int> & vList)
{
    int i;
    for( i = 0; i < vList.size(); i++)
    {
        int n = rand() % 100;
        vList[i]= n;
    }
}
```



Vector como valor retornado

- Podemos declarar un vector dentro de una función y retornar una copia de él:

```
vector<int> makeRandomVector( int count )  
{  
    vector<int> list(count); // set the size  
    int i;  
  
    for( i = 0; i < count; i++) {  
        list[i] = rand();  
    }  
  
    return list; // return a copy  
}
```

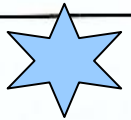
- `vector<int> Z = makeRandomVector(25);`

Algoritmos Estándares útiles en Vectores

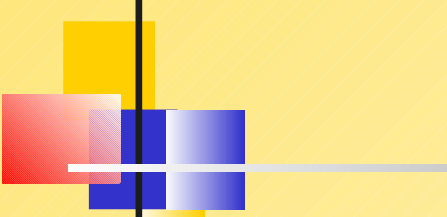


- Encontrar un valor único
- Contar el número de coincidencias
- Recolectar todos los valores coincidentes
- Remover un elemento
- Insertar un elemento

Table 8.2 Generic algorithms useful with vectors



<code>fill (iterator start, iterator stop, value)</code>	Fill vector with a given initial value
<code>copy (iterator start, iterator stop, iterator destination)</code>	Copy one sequence into another
<code>max_element(iterator start, iterator stop)</code>	Find largest value in collection
<code>min_element(iterator start, iterator stop)</code>	Find smallest value in collection
<code>reverse (iterator start, iterator stop)</code>	Reverse elements in the collection
<code>count (iterator start, iterator stop, target value, counter)</code>	Count elements that match target value, incrementing counter
<code>count_if (iterator start, iterator stop, unary fun, counter)</code>	Count elements that satisfy function, incrementing counter
<code>transform (iterator start, iterator stop, iterator destination, unary)</code>	Transform elements using unary function from source, placing into destination
<code>find (iterator start, iterator stop, value)</code>	Find value in collection, returning iterator for location
<code>find_if (iterator start, iterator stop, unary function)</code>	Find value for which function is true, returning iterator for location
<code>replace (iterator start, iterator stop, target value, replacement value)</code>	Replace target element with replacement value
<code>replace_if (iterator start, iterator stop, unary fun, replacement value)</code>	Replace elements for which fun is true with replacement value
<code>sort (iterator start, iterator stop)</code>	Places elements into ascending order
<code>for_each (iterator start, iterator stop, function)</code>	Execute function on each element of vector
<code>iter_swap (iterator, iterator)</code>	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

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



Búsqueda de la primera coincidencia

- Retorna el índice del primer valor en el vector que coincida con un valor dado (o retorna -1)

```
int findMatch(const vector<int> & vec, int t)
{
    int i = 0;
    while (i < vec.size())
    { if(vec[i] == t)
        return i;
      else
        i++;
    }
    return -1;    / no match was found
}
```



Cuenta el Número de Coincidencias

- ¿Cuántos valores son mayores que un valor dado?

```
int count_greater( const vector<double> & vec, double cutoff )
{
    int count = 0;
    int i;
    for (i = 0; i < vec.size(); i++)
    {
        if( vec[i] > cutoff )
            count++;
    }
    return count;
} // source: Horstmann
```



Recolección de Coincidencias

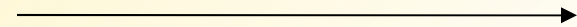
- Encuentra y retorna las posiciones de todos los valores superiores a un umbral dado.

```
vector<int> find_all_greater(  
    const vector<double> & vec, double t)
```

PURPOSE: Find all values in a vector that
are greater than a threshold

RECEIVES: vec - the vector
t - the threshold value

RETURNS: a vector of the positions of
all values that are greater
than the value t





Recolección de Coincidencias

```
vector<int> find_all_greater(  
    const vector<double> & vec, double t)  
{  
    vector<int> pos;  
    int i;  
    for (i = 0; i < vec.size(); i++)  
    {  
        if( vec[i] > t )  
            pos.push_back(i);  
    }  
    return pos;  
}  
  
// source: Cay Horstmann
```



Eliminación de un Elemento

- Si el orden no es importante, sobre-escribir el elemento removido con el último elemento.

```
void erase(vector<string>& vec, int pos)
{
    int last_pos = vec.size() - 1;
    vec[pos] = vec[last_pos];
    vec.pop_back();
}
// Source: Cay Horstmann
```



Eliminación de un Elemento

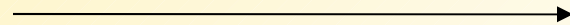
- Si el orden es importante, mover hacia abajo todos los elementos sobre el elemento removido.

```
void erase(vector<string>& vec, int pos)
{
    int i;
    for (i = pos; i < vec.size() - 1; i++)
        vec[i] = vec[i + 1];
    vec.pop_back(); // remove last elt
}
// Source: Cay Horstmann
```



Inserción de un Elemento

- La inserción de un nuevo elemento en el medio de un vector ordenado
- Algoritmo
 - Duplicar el último elemento
 - Comenzar desde atrás, correr cada elemento hacia atrás en una posición hasta alcanzar la posición de inserción
 - Insertar el nuevo elemento en el espacio abierto





Inserción de un Elemento

```
void insert(vector<string> & vec,
            int pos, string s)
{
    int last = vec.size() - 1;

    // duplicate the last element
    vec.push_back( vec[last] );

    // slide elements back to open a slot
    int i;
    for (i = last; i > pos; i--)
        vec[i] = vec[i - 1];

    vec[pos] = s; // insert the new element
}
```




Ordenar un Vector

- Asumir que el vector contiene ítems cuyos tipos/clases es predefinido en C++
- La función `sort()` pertenece a la biblioteca `<algorithm>`
- `begin()` apunta al primer elemento, y `end()` apunta al posición siguiente al último elemento
 - `#include <algorithm>`
 - `vector<int> items;`
 - `sort(items.begin(), items.end());`



Iteradores (Iterators)

- Un *iterador* es un puntero a un elemento de un vector que puede movido hacia delante o hacia atrás a través de los elementos del vector.
- *Desreferenciamos* un iterador para acceder los elementos que este 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 number`



Ordenamiento Usando Iteradores

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

```
#include <algorithm>
```

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

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

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

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

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

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

Ordenamiento de 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> cop3337;
```

```
sort( cop3337.begin(), cop3337.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 );
```