

Vectors (Vectores)

Agustin J. González
Versión Original de Kip Irvine
ELO320: Seminario II
2do. Sem 2001

Copyright Kip Irvine 2001, all rights reserved.

Contenidos

- Qué es un vector?
- Declaración de Objetos Vectores
- Inserción y eliminación de items
- Uso de sub-índices
- Obtención y modificación del tamaño
- Preasignación 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 más simple de las clases contenedoras de la STL y en muchos casos la más eficiente.

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 Ítems

- `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:

$$0 \leq n < \text{size}()$$

- 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 in 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 in 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 in 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 in 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 in 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 el 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 como valor retornado

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

Algoritmos Estándares de Vectores

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

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.
- *Dereferenciamos* un iterador para acceder los elementos que este apunta. (* = operador de dereferencia)

```
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 `sort()`

```
#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 );
```

Fin

