

STL Colecciones

Antonio Espín Herranz

STL: Las clases Contenedoras

- Referencia a la STL
<http://www.cplusplus.com/reference/>
- Dentro de la **STL** (La biblioteca estándar de patrones) tenemos definidos una serie de clases contenedoras.
- Estas clases nos permiten contener a otros objetos.
- C++ también suministra una serie de iteradores para poder recorrer estos contenedores.
- Los mas comunes son **vector** y **string**.

Tipos de contenedores

- **Secuenciales:**
 - **Vectores:** contienen elementos contiguos almacenados al estilo de un array o vector del lenguaje C++. **<vector>**
 - **Listas:** secuencias de elementos almacenados en una lista enlazada. **<list>**
 - **Deque:** contenedores parecidos a los vectores, excepto que permiten inserciones y borrados tanto al principio como al final. **<deque>**
- **Adaptadores:**
 - **Colas:** contenedores que ofrecen la funcionalidad de listas "primero en entrar, primero en salir". **<queue>**
 - **Pilas:** contenedores asociados a listas "primero en entrar, último en salir". **<stack>**
 - **Colas con prioridad:** en esta caso, los elementos de la cola salen de ella de acuerdo con una prioridad (que se estableció en la inserción). **<priority_queue>**
- **Asociativos.**
 - **Conjuntos de bits:** contenedor para almacenar bits. **<bitset>**
 - **Mapas:** almacenan pares "clave, objeto", es decir, almacenan objetos referidos mediante un identificador único. **<map>**
 - **Multimapas:** mapas que permiten claves duplicadas. **<multimap>**
 - **Conjuntos:** conjuntos ordenados de objetos únicos. **<set>**
 - **Multiconjuntos:** conjuntos ordenados de objetos que pueden estar duplicados. **<multiset>**

Vector

- El contenedor vector permite almacenar cero o más objetos del mismo tipo.
- Permite acceder a ellos individualmente mediante un índice, es decir, acceso aleatorio.
- En este sentido, es una extensión del vector o array que ofrece C++, aunque en este caso el número de elementos de un objeto vector puede variar dinámicamente.
- La gestión de la memoria se hace de manera totalmente transparente al usuario.
- Se define como una clase patrón, lo que implica que puede albergar objetos de cualquier tipo.
- En cuanto a las operaciones más frecuentes, ofrece un tiempo constante en inserción y borrados de elementos al final, y lineal al comienzo o en la mitad del vector.

Vector

- Está definido en el fichero <vector>

- Ejemplos para definir un vector:

- *vector<tipo> objeto;*

- vector<double> vectorReales; // De números reales.*

- vector<string> vectorCadenas; // De cadenas de caracteres.*

- vector<MiClase> vectorObj; // Contendrá objetos de una clase construida por un usuario.*

- También podemos indicar la dimensión y un valor por defecto.

- *vector <int> vectorEnteros(10 [, valor_defecto])*

- O indicar el valor por defecto: *vector <int>vectorEnteros(10, -1);*

- Puede contener objetos de cualquier tipo, tanto predefinido como definido por el usuario.

Miembros de vector

- **size** size_type size() const;
Devuelve el número de elementos almacenados en el vector. El tipo `size_type` es un entero sin signo.
- **empty** bool empty() const;
Devuelve true si el número de elementos es cero y false en caso contrario.
- **push_back** void push_back(const T& x);
Añade un elemento x al final del vector. T es el tipo de dato de los elementos del vector.

```
vector<int> a;
```

```
a.push_back(5);
```

- **begin** iterator begin();
Devuelve un iterador que referencia el comienzo del vector.
- **end** iterator end();
Devuelve un iterador que referencia la posición siguiente al final del vector.

Miembros de vector II

- **erase** `void erase(iterator first, iterator last);`
Borra los elementos del vector que estén situados entre los iteradores first y last.

```
vector<int> a;
```

```
...
```

```
a.erase(a.begin(),a.end()); // Se borran todos los  
elementos entre la primera y la última posición.
```

- **capacity** `size_type capacity() const`
Devuelve el número de elementos con que se ha creado el vector. Siempre es mayor o igual que size.

-

clear `void clear ();`
Borra todos los elementos de un vector.

```
vector<int> a;
```

```
a.clear(); // Se borran todos los elementos.
```

Miembros de vector III

- **front()** y **back()** nos devuelven el primer y el último elemento.
- **resize(nuevo_tamaño)**: Redimensiona el vector.
- **pop_back()**: Nos permite eliminar el último elemento del vector. Con `erase` eliminamos un rango.
- **Operadores de relación**: También se puede aplicar entre objetos vector. `<`, `<=`, `>`, `>=`, `==`, `!=`
- **Acceso a los elementos de un vector**: mediante:
 - `vector<int> v;`
 - `v.at(i)` // Verifica el rango `v[i]` // Sin verificación del rango.

Miembros de vector IV

- El método **at** si no existe la posición lanzará una excepción: **out_of_range**
- El acceso con el **[]**, no verifica el rango y puede que el **programa falle o invada una zona de memoria no reservada.**
- Cuando cargamos un vector podemos utilizar **push_back** para hacerlo en un bucle o el método **insert** (a continuación).

Miembros de vector V

- Método **insert**: podemos insertar un elemento dentro de un vector en cualquier posición.

- Indicamos a partir de que posición, el número de elementos y el valor.

```
k=17;
```

```
v.push_back(k); // Añade por el final.
```

```
v.insert(v.begin()+3,2,0) // Añade dos elementos con valor inicial 0  
a partir de v[3].
```

- *El método **insert** es bastante potente para **cargar en un vector los elementos de un array** y hacerlo de un plumazo sin tener que escribir un bucle, utiliza aritmética de punteros para indicar el inicio y fin del array:*

```
void cargarVector(vector<int> &v, int *p, int n){  
    v.insert(v.begin(), p, p+n);  
}
```

Miembros de vector VI

- Podemos utilizar algoritmos de búsqueda definidos en *<algorithm>*.
- **find** indicamos entre que región queremos buscar y el valor a buscar.

Iteradores

- Para poder recorrer un vector necesitamos iteradores:
- Definición:

```
vector<int> v(20);  
vector<int>::iterator e;  
Vector<int>::reverse_iterator e1;
```

```
// Para recorrer el vector de inicio a fin:  
for (e = v.begin() ; e != v.end() ; e++)
```

```
// Para recorrer el vector del fin a inicio:  
for (e1 = v.rbegin() ; e1 != rend() ; e1++ )
```

Importante

- Cuando definimos un vector de una clase nuestra y utilizamos los operadores de vector como `==` u otro operador relacional estos tienen que estar implementados en la clase.

Por ejemplo

```
vector<MiClase> v1, v2;
```

```
...
```

```
if (v1 == v2)
```

```
    // C++ buscará el operador == en MiClase
```

```
    // Implementarlo con una función friend.
```

Map

- Representa un contenedor asociativo.
- Está formado por pares de clave / valor.
- Se incluye en el fichero <map>.
- Sintaxis:
map <tipo1, tipo2> nombreObj;
- El primer elemento del par representa la clave y se utiliza para localizar el 2º elemento.

Miembros de map

- Proporciona métodos similares a lo de vector, iteradores, algoritmos, etc.
- first y second para poder acceder al primer elemento y al segundo.
- erase, clear, size, empty.
- Para añadir elementos al mapa:
`miMapa[clave]=valor`

string

- Clase para trabajar con cadenas de caracteres de una forma mas sencilla que con `char *`.
- Se incluye en el fichero: `<string>`

Miembros de string

- *Constructores*

- `string s;` Constructor por defecto
- `string s (“hola”);` Constructor con inicializador.
- `string s = “hola”;`
- `string s (aString);` Constructor de copia

- *Acceso a elementos*

- `s[i];` Acceso al elemento i-ésimo del string
- `s.substr(int pos,int len);` Subcadena que comienza en pos y tiene longitud len.
- `s.c_str();` Devuelve una cadena estilo C igual al string

Miembros de string II

- *Insertión y borrado:*
 - `s.insert(int pos,string str);` Insertar antes de pos el string str
 - `s.erase (int start, int len);` Eliminar desde `s[start]` hasta `s[start+len]`
 - `s.replace(int start, int len,str);` Sustituir desde `s[start]` hasta `s[start+len]` por str
- *Longitud*
 - `s.length();` Longitud del string
 - `s.resize(int,char);` Cambia el tamaño, rellenando con un valor.
 - `s.empty();` Cierto si el string es vacío.

Miembros de string III

- **Asignación**

- `s = s2;` Asignación de strings.
- `s += s2;` Concatenación de strings.
- `otroString = s + s2;` Nuevo string resultado de concatenar `s` y `s2`.

- **Comparaciones**

- `s == s2;` `s != s2;` Igualdad y desigualdad de strings.
- `s < s2;` `s <= s2;` Comparaciones de strings (orden lexicográfico).
- `s > s2` `s >= s2;` Comparaciones de strings (orden lexicográfico)

- **Iteradores**

- `string::iterator s;` Declara un nuevo iterador.
- `s.begin ();` Iterador que referencia al primer elemento
- `s.end ();` Iterador que referencia al al último.
- `string::reverse_iterator s;` Declara un nuevo `reverse_iterator`.
- `s.rbegin ();` `Reverse_iterator` que referencia al último elemento
- `s.rend ();` `Reverse_iterator` que referencia al anterior al primero

Miembros de string IV

- ***Operaciones de búsqueda***

- s.find(string str, int pos); Devuelve la posición en donde comienza la subcadena str desde s[pos].
- s.find_first_of(str,pos); Posición en donde se encuentra el primer carácter que pertenece a str desde s[pos].
- s.find_first_not_of(str,pos); Posición en donde se encuentra el primer carácter que no está en str desde s[pos].
- s.find_last_of(str,pos); Posición en donde se encuentra el último carácter que pertenece a str desde s[pos].
- s.find_last_not_of(str,pos) Posición en donde se encuentra el último carácter que no está en str desde s[pos].

- ***Operaciones E/S***

- stream >> strEntrada de string
- sstream << strSalida de strings
- getline(stream,str,char); Añade a str todos los caracteres de una línea de la entrada estándar hasta encontrar el carácter char. Por defecto char es igual a '\n'.