UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y CC. Computación
E.T.S.I. Telecomunicación

CONTENEDORES DE LA BIBLIOTECA ESTÁNDAR

Tema 5

Programación II

Tema 5: CONTENEDORES DE LA BIBLIOTECA ESTÁNDAR

1. Introducción
2. El Tipo Vector
3. Vectores 2D
4. El TAD Pila
5. El TAD Cola
6. Bibliografía

INTRODUCCIÓN (I)

- Los *contenedores* de la biblioteca estándar (*STL*) proporcionan un método general para almacenar y acceder a una colección de elementos homogéneos
- Proporcionan diferentes características adaptadas a diferentes necesidades
- En este tema se mostrarán las principales operaciones que se pueden realizar con el contenedor de la biblioteca estándar `vector`.
- Además, implementaremos los TADs Pila y Cola utilizando para ello el tipo `vector`.
- Son **tipos compuestos**, por lo tanto, el paso de parámetros:
  - Los parámetros de entrada se pasarán por referencia constante
  - Los parámetros de salida y entrada/salida se pasarán por referencia
  - Su devolución desde una función está desaconsejada

INTRODUCCIÓN (II)

- La biblioteca estándar también define otros tipos de contenedores optimizados para diferentes circunstancias.
- Definidos dentro del espacio de nombres **std**
- En *GNU G++*, la opción de compilación `-D_GLIBCXX_DEBUG` permite comprobar los índices de acceso.

Contenedor	**Tipo**	**Acceso**	**Inserción**	**Eliminación**
`stack` (adaptador)	*TAD Pila*	Directo (al final)	Al final	Al final
`queue` (adaptador)	*TAD Cola*	Directo (al principio)	Al final	Al principio
`array`	Secuencia	Directo (pos)	–	–
`vector`	Secuencia	Directo (pos)	Al final	Al final
`deque`	Secuencia	Directo (pos)	Al final + al principio	Al final + Al principio
`list`	Secuencia	Secuencial (bidir)	Cualquier posición	Cualquier posición
`forward_list`	Secuencia	Secuencial (fw)	Cualquier posición	Cualquier posición
`map`	Asociativo	Binario por clave	Por Clave	Por Clave
`set`	Asociativo	Binario por clave	Por Clave	Por Clave
`multimap`	Asociativo	Binario por clave	Por Clave	Por Clave
`multiset`	Asociativo	Binario por clave	Por Clave	Por Clave
`unordered_map`	Asociativo	Hash por clave	Por Clave	Por Clave
`unordered_set`	Asociativo	Hash por clave	Por Clave	Por Clave
`unordered_multimap`	Asociativo	Hash por clave	Por Clave	Por Clave
`unordered_multiset`	Asociativo	Hash por clave	Por Clave	Por Clave

Tema 5: CONTENEDORES DE LA BIBLIOTECA ESTÁNDAR

1. Introducción
2. El Tipo Vector
3. Vectores 2D
4. El TAD Pila
5. El TAD Cola
6. Bibliografía

EL TIPO VECTOR (I)

- El contenedor de tipo `vector` representa una secuencia de elementos homogéneos (*genéricos*) optimizada para:
  - El acceso directo a los elementos según su posición
  - La inserción de elementos al final de la secuencia
  - La eliminación de elementos del final de la secuencia
- Fichero de interfaz `<vector>`, espacio de nombres **std**
- Similar al tipo `array`, pero su tamaño puede crecer en tiempo de ejecución dependiendo de las necesidades surgidas durante la ejecución del programa.

EL TIPO VECTOR (II)

- El constructor por defecto de un *vector* crea un objeto vector vacío (sin elementos almacenados).
- El constructor de copia de un *vector* crea un nuevo objeto vector con tantos elementos como el vector original, y posteriormente invoca al constructor de copia para copiar cada elemento nuevo desde el elemento del vector original.
- El destructor de un *vector* invoca al destructor de cada elemento almacenado, y posteriormente destruye el propio vector.
- El comportamiento del operador de asignación de un *vector* es similar a destruir el vector actual e invocar al constructor de copia con el nuevo vector original a copiar, aunque en ocasiones puede resolverse simplemente invocando al destructor, constructor por defecto, constructor de copia y operador de asignación.
- El constructor específico con tamaño de un *vector* crea un objeto vector con tantos elementos como se especifica en el parámetro, y se invoca al constructor por defecto sobre cada nuevo elemento almacenado.
- El constructor específico con tamaño y valor de un *vector* crea un objeto vector con tantos elementos como se especifica en el parámetro, y se invoca al constructor de copia para copiar el valor especificado sobre cada nuevo elemento almacenado.

EL TIPO VECTOR (III)

```
#include <vector>
using namespace std;
typedef vector<int> VectorInt;                  // Instanciación del tipo vector
int main()
{
    vector<int> v1;                             // { }            Ctor por defecto: vector vacío
    VectorInt v2(4);                            // { 0 0 0 0 }    Ctor 4 elementos valor por defecto
    VectorInt v3(4, 5);                         // { 5 5 5 5 }    Ctor 4 elementos valor inicial 5
    VectorInt v4 = v3;                          // { 5 5 5 5 }    Ctor copia de v3
    if (v4 <= v3) {                             // Comparación lexicográfica de vectores
        v4 = VectorInt(4);                      // Asignar vector 4 elementos con valor por defecto
        v4.swap(v3);                            // Intercambio eficiente de vectores
    }
    for (int i = 0; i < int(v4.size()); ++i) {  // Iteración sobre el número de elementos del vector
        v4[i] = v4[i] * 2;                      // Acceso al elemento de la posición (i) del vector
    }
    v4.clear();                                 // { }     Eliminar el contenido del vector (pasa a vacío)
    v4.push_back(1);                            // { 1 }   Añadir un elemento al final de la secuencia
    v4.push_back(2);                            // { 1 2 } Añadir un elemento al final de la secuencia
    v4.resize(5, 7);                            // { 1 2 7 7 7 } Cambia el número de elementos de la sec.
    v4.resize(3);                               // { 1 2 7 }     Cambia el número de elementos de la sec.
    if ( ! v4.empty() ) {                       // Comprueba si el vector está vacío (v4.size() == 0)
        v4.pop_back();                          // { 1 2 }       Elimina el último elemento de la sec.
    }
}                                               // Destruye los elementos y el propio vector
```

EL TIPO VECTOR (IV)

- En caso de utilizar el contenedor de tipo `vector` como tipo del **atributo** de una **clase**, entonces debemos considerar su construcción en la **lista de inicialización**.

```
#include <vector>
namespace umalcc {
    class Datos {
    public:
        Datos();
        Datos(int nm);
        Datos(const Datos& o);
        Datos& operator=(const Datos& o);
    private:
        typedef std::vector<double> VDouble;
        //-- Atributos --
        VDouble muestra;  // Equivalente a: std::vector<double> muestra;
    };
}
namespace umalcc {
    Datos::Datos()        // Constructor por defecto
      : muestra() {}      //  Invoca al constructor por defecto sobre el atributo de tipo vector
    Datos::Datos(int nm)  // Constructor específico
      : muestra(nm, 0) {} // Invoca al ctor. específico (tamaño, valor) sobre el atributo de tipo vector
    Datos::Datos(const Datos& o)  // Constructor de copia
      : muestra(o.muestra) {}     // Invoca al ctor. de copia sobre el atributo de tipo vector
    Datos& Datos::operator=(const Datos& o)                    // Operador de asignacion
      { if (this != &o) { muestra = o.muestra; } return *this; } // Invoca al op. de asignacion de vector
}
```

EL TIPO VECTOR (V)

- Atención cuando se devuelve un vector como parámetro de salida.
- Se debe **vaciar** (con `clear()`) antes de añadir información al vector de salida.
- Ejemplo: seleccionar y devolver en un nuevo vector `v2` todos los elementos de `v1` que sean mayores que `limite`.

```
#include <vector>
using namespace std;
typedef vector<int> VInt;

void seleccionar(const VInt& v1, int limite, VInt& v2)
{
    v2.clear();
    for (int i = 0; i < int(v1.size()); ++i) {
        if (v1[i] > limite) {
            v2.push_back(v1[i]);
        }
    }
}
```

Tema 5: CONTENEDORES DE LA BIBLIOTECA ESTÁNDAR

1. Introducción
2. El Tipo Vector
3. Vectores 2D
4. El TAD Pila
5. El TAD Cola
6. Bibliografía

```
#include <vector>
using namespace std;
typedef vector<int> FilaInt;                    // Fila de int
typedef vector<FilaInt> MatrizInt;              // Matriz de int (vector de Filas)
int main()
{
    MatrizInt m1;                               // Matriz 0 filas X 0 columnas
    MatrizInt m2(2, FilaInt(3));                // Matriz 2 filas X 3 columnas con valor por defecto
    MatrizInt m3(2, FilaInt(3, 5));             // Matriz 2 filas X 3 columnas con valor inicial 5
    MatrizInt m4 = m3;                          // Ctor de copia de m3
    if (m4 <= m3) {                             // Comparación lexicográfica de matrices
        m4 = MatrizInt(5, FilaInt(7));          // Asignar matriz 5 X 7 elementos con valor por defecto
        m4.swap(m3);                            // Intercambio eficiente de matrices
    }
    for (int f = 0; f < int(m4.size()); ++f) {  // Iteración sobre las filas de la matriz
        for (int c = 0; c < int(m4[f].size()); ++c) {  // Iteración sobre las columnas de la fila f
            m4[f][c] = m4[f][c] * 2;            // Acceso al elemento de la posición (f,c) de la matriz
        }
    }
    m4.clear();                                 // Eliminar el contenido de la matriz (pasa a vacío)
    for (int f = 0; f < 3; ++f) {
        m4.push_back(FilaInt());                // Añadir una fila vacia al final de las filas
        for (int c = 0; c < 5; ++c) {
            m4[m4.size()-1].push_back(0);       // Añadir un elemento al final en la última fila
        }
    }
    m4.resize(9, FilaInt(5, 2));                // Cambia el número de filas a 9 (añade filas 5 elem val 2)
    m4.resize(7);                               // Cambia el número de filas a 7
    if ( ! m4.empty() ) {                       // Comprueba si la matriz está vacía (v4.size() == 0)
        m4.pop_back();                          // Elimina la última fila de la matriz
    }
}
```

Tema 5: CONTENEDORES DE LA BIBLIOTECA ESTÁNDAR

1. Introducción
2. El Tipo Vector
3. Vectores 2D
4. El TAD Pila
5. El TAD Cola
6. Bibliografía

EL TAD PILA. DEFINICIÓN

- Colección ordenada (según el orden de inserción) de elementos homogéneos donde se pueden introducir elementos (manteniendo el orden de inserción) y sacar elementos de ella (en orden inverso al orden de inserción), de tal forma que el **primer** elemento que sale de la pila es el **último** elemento que ha sido introducido en ella.

- El TAD Pila proporciona las siguientes operaciones públicas:
  - Constructor por defecto: construye un objeto *Pila* vacío.
  - Constructor de copia: construye un objeto *Pila* copiando sus valores de otro objeto *Pila*.
  - Operador de asignación: asigna a un objeto *Pila* el valor de otro objeto *Pila*.
  - Añadir un elemento a la **cima** del objeto *Pila*.
  - Acceder al valor del elemento que se encuentra en la **cima** del objeto *Pila*. Esta operación requiere que la pila no esté vacía.
  - Eliminar el elemento que se encuentra en la **cima** del objeto *Pila*. Esta operación requiere que la pila no esté vacía.
  - Comprobar si el objeto *Pila* está vacío.
  - Destructor: destruye el objeto *Pila* y todos sus recursos asociados.

EL TAD PILA. IMPLEMENTACIÓN CON VECTOR

```
#ifndef pila_hpp_
#define pila_hpp_
#include <vector>
namespace umalcc {
    class Pila {
    public:
        typedef int TipoElem;
        ~Pila();
        Pila();
        Pila(const Pila& o);
        Pila& operator=(const Pila& o);
        void push(const TipoElem& val);
        TipoElem top() const;
        void pop();
        bool empty() const;
    private:
        std::vector<TipoElem> v;
    } ;
}
#endif
```

```
#include "pila.hpp"
namespace umalcc {
    Pila::~Pila() {}
    Pila::Pila() : v() {}
    Pila::Pila(const Pila& o) : v(o.v) {}
    Pila& Pila::operator=(const Pila& o) {
        v = o.v;
        return *this;
    }
    void Pila::push(const TipoElem& val) {
        v.push_back(val);
    }
    Pila::TipoElem Pila::top() const {
        TipoElem r = TipoElem();
        if (v.size() > 0) {
            r = v[v.size()-1];
        }
        return r;
    }
    void Pila::pop() {
        if (v.size() > 0) {
            v.pop_back();
        }
    }
    bool Pila::empty() const {
        return v.size() == 0;
    }
}
```

Tema 5: CONTENEDORES DE LA BIBLIOTECA ESTÁNDAR

1. Introducción
2. El Tipo Vector
3. Vectores 2D
4. El TAD Pila
5. El TAD Cola
6. Bibliografía

EL TAD COLA. DEFINICIÓN

- Colección ordenada (según el orden de inserción) de elementos homogéneos donde se pueden introducir elementos (manteniendo el orden de inserción) y sacar elementos de ella (en el mismo orden al orden de inserción), de tal forma que el **primer** elemento que sale de la cola es el **primer** elemento que ha sido introducido en ella.

- El TAD Cola proporciona las siguientes operaciones públicas:
  - Constructor por defecto: construye un objeto *Cola* vacío.
  - Constructor de copia: construye un objeto *Cola* copiando sus valores de otro objeto *Cola*.
  - Operador de asignación: asigna a un objeto *Cola* el valor de otro objeto *Cola*.
  - Añadir un elemento al **final** del objeto *Cola*.
  - Acceder al valor del elemento que se encuentra al **principio** del objeto *Cola*. Esta operación requiere que la cola no esté vacía.
  - Eliminar el elemento que se encuentra al **principio** del objeto *Cola*. Esta operación requiere que la cola no esté vacía.
  - Comprobar si el objeto *Cola* está vacío.
  - Destructor: destruye el objeto *Cola* y todos sus recursos asociados.

EL TAD COLA. IMPLEMENTACIÓN SIMPLE CON VECTOR (I)

```
#ifndef cola_hpp_
#define cola_hpp_
#include <vector>
namespace umalcc {
    class Cola {
    public:
        typedef int TipoElem;
        ~Cola();
        Cola();
        Cola(const Cola& o);
        Cola& operator=(const Cola& o);
        void push(const TipoElem& val);
        TipoElem front() const;
        void pop();
        bool empty() const;
    private:
        std::vector<TipoElem> v;
    } ;
}
#endif
//---------------------------------------
#include "cola.hpp"
namespace umalcc {
    Cola::~Cola() {}
    Cola::Cola() : v() {}
    Cola::Cola(const Cola& o) : v(o.v) {}
    Cola& Cola::operator=(const Cola& o) {
        v = o.v;
        return *this;
    }
    void Cola::push(const TipoElem& val) {
        v.push_back(val);
    }
    Cola::TipoElem Cola::front() const {
        TipoElem r = TipoElem();
        if (v.size() > 0) {
            r = v[0];
        }
        return r;
    }
    void Cola::pop() {
        if (v.size() > 0) {
            for (unsigned i = 0; i < v.size()-1; ++i) {
                v[i] = v[i+1];
            }
            v.pop_back();
        }
    }
    bool Cola::empty() const {
        return v.size() == 0;
    }
}
```

EL TAD COLA. IMPLEMENTACIÓN MÁS EFICIENTE

- A continuación se muestra otra implementación alternativa **más eficiente** del *TAD Cola*, ya que en la implementación anterior, cada vez que se extrae un elemento se deben desplazar los elementos hacia el principio. En esta nueva implementación no es necesario el desplazamiento de los elementos, salvo cuando de debe duplicar el tamaño del vector cuando esta lleno.

EL TAD COLA. IMPLEMENTACIÓN CON VECTOR (I)

```
#ifndef cola_hpp_
#define cola_hpp_
#include <vector>
namespace umalcc {
    class Cola {
    public:
        typedef int TipoElem;
        ~Cola();
        Cola();
        Cola(const Cola& o);
        Cola& operator=(const Cola& o);
        void push(const TipoElem& val);
        TipoElem front() const;
        void pop();
        bool empty() const;
    private:
        std::vector<TipoElem> v;
        unsigned nelms;
        unsigned ini;
        unsigned fin;
    } ;
}
#endif
```

EL TAD COLA. IMPLEMENTACIÓN CON VECTOR (II)

```
#include "cola.hpp"
namespace umalcc {
    Cola::~Cola() {}
    Cola::Cola()
        : v(32), nelms(0), ini(0), fin(0) {}
    Cola::Cola(const Cola& o)
        : v(o.v), nelms(o.nelms),
          ini(o.ini), fin(o.fin) {}
    Cola& Cola::operator=(const Cola& o) {
        v = o.v; nelms = o.nelms;
        ini = o.ini; fin = o.fin;
        return *this;
    }
    Cola::TipoElem Cola::front() const {
        TipoElem r = TipoElem();
        if (nelms > 0) {
            r = v[ini];
        }
        return r;
    }
    void Cola::push(const TipoElem& val) {
        if (nelms == v.size()) {
            v.resize(2 * v.size());
            for (unsigned i = 0; i < fin; ++i) {
                v[nelms+i] = v[i];
            }
            fin += nelms;
        }
        v[fin] = val;
        fin = (fin + 1) % v.size();
        ++nelms;
    }
    void Cola::pop() {
        if (nelms > 0) {
            ini = (ini + 1) % v.size();
            --nelms;
        }
    }
    bool Cola::empty() const {
        return nelms == 0;
    }
}
```

Tema 5: CONTENEDORES DE LA BIBLIOTECA ESTÁNDAR

1. Introducción
2. El Tipo Vector
3. Vectores 2D
4. El TAD Pila
5. El TAD Cola
6. Bibliografía