

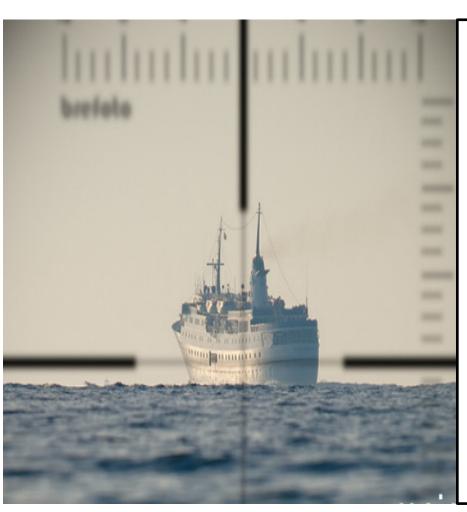
## Listas



Programación III.

uso

# Objetivos



- Conocer el modelo del tipo abstracto de datos de STL
- Elaborar ejercicios de aplicación de STL

**USO** 

- La clase list es una clase template (plantilla) en la Biblioteca estándar C++\*
- Podemos crear listas que contengan cualquier tipo de objeto.
- Las clases list y vector comparten muchas operaciones, incluyendo: push back(), pop back(), begin(), end(), size(), y empty()
- EL operador sub-índice ([]) no puede ser usado con listas.

<sup>\*</sup> Esta no es exactamente la misma que la "Standard Template Library" (STL) actualmente mantenida por Silicon Graphics Corporation (www.sgi.com), pero compatible en la gran mayoría de los casos.

### Agregar y remover nodos

El siguiente código crea una lista, agrega cuatro nodos, y remueve un nodo:

```
#include <list>
list <string> staff;
staff.push back("Fred");
staff.push back("Jim");
staff.push back("Anne");
staff.push back("Susan");
cout << staff.size() << endl;</pre>
staff.pop back();
cout << staff.size() << endl;</pre>
```

- Un iterador (iterator) es un puntero que se puede mover a través de la lista y provee acceso a elementos individuales.
- El operador referencia (\*) es usado cuando necesitamos obtener o fijar el valor de un elemento de la lista.

Podemos usar los operadores ++ y -- para manipular iteradores. El siguiente código recorre la lista y despliega los ítems usando un iterador:

```
void ShowList( list<string> & sList )
  list<string>::iterator pos;
  pos = sList.begin();
  while( pos != sList.end())
    cout << *pos << endl;</pre>
    pos++;
```

Si pasmos una lista como constante (const list) debemos usar un iterador constante para recorrer la lista:

```
void ShowList( const list<string> & sList )
  list<string>::const iterator pos;
  pos = sList.begin();
  while( pos != sList.end())
    cout << *pos << endl;</pre>
    pos++;
```



# Iterador reverso (reverse

Un iterador reverso (reverse iterator) recorre la lista en dirección inversa. EL siguiente bucle despliega todos los elementos en orden inverso:

```
void ShowReverse( list<string> & sList )
  list<string>::reverse iterator pos;
  pos = sList.rbegin();
  while( pos != sList.rend())
    cout << *pos << endl;</pre>
    pos++;
```



#### int main(

Facultad de Ingeniería y Ciencias Naturales

#### (const\_reverse\_iterátör)

Un const\_reverse\_iterator nos permite trabajar con objetos lista constantes:

```
void ShowReverse( const list<string> & sList )
  list<string>::const reverse iterator pos;
  pos = sList.rbegin();
  while( pos != sList.rend())
    cout << *pos << endl;</pre>
    pos++;
```

La función miembro insert() inserta un nuevo nodo antes de la posición del

```
list<string> staff;
staff.push back("Barry");
staff.push back("Charles");
list<string>::iterator pos;
pos = staff.begin();
staff.insert(pos, "Adele");
// "Adele", "Barry", "Charles"
pos = staff.end();
staff.insert(pos, "Zeke");
// "Adele", "Barry", "Charles", "Zeke"
```

iterador. EL iterador sigue siendo válido después de la operación.

La función miembro erase() remueve el nodo de la posición del iterador. El iterador es no válido después de la operación.

```
list<string> staff;
staff.push back("Barry");
staff.push back("Charles");
list<string>::iterator pos = staff.begin();
staff.erase(pos);
cout << *pos;</pre>
                        // error:invalidated!
// erase all elements
staff.erase( staff.begin(), staff.end());
cout << staff.empty(); // true</pre>
```

La función miembro merge() combina dos listas en según el operador de orden de los objetos que contiene. Por ejemplo en este caso el orden es alfabético.

```
list <string> staff1;
staff1.push back("Anne");
staff1.push back("Fred");
staff1.push back("Jim");
staff1.push back("Susan");
list <string> staff2;
staff2.push back("Barry");
staff2.push back("Charles");
staff2.push back("George");
staff2.push back("Ted");
staff2.merge( staff1 );
```

@

La función miembro sort() ordena la lista en orden ascendente. La función reverse() invierte la lista.

```
list <string> staff;
.
.
staff.sort();
staff.reverse();
```

