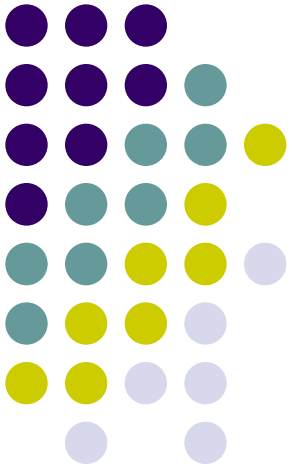
http://webs.um.es/frgarcia/teaching.htm

ALGORITMOS Y ESTRUCTURAS DE DATOS



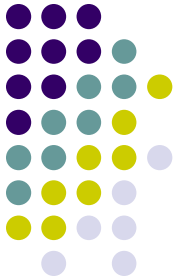
PRÁCTICA. TEMAS 2 Y 3

SEMINARIO DE C++

SESIÓN 3

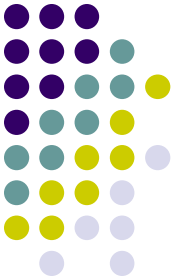
Francisco García Sánchez
frgarcia@um.es
Despacho 2.31

INDICE DE CONTENIDOS



- Introducción a C++
  - Funciones y clases genéricas
  - Excepciones
  - Asertos
  - El puntero '`this`'
  - Redefinición de operadores
- Introducción a la STL en C++
- Planificación práctica

Descargar: http://dis.um.es/~ginesgm/files/doc/aed/seminario2-3.pdf



Introducción a C++

- Funciones y clases genéricas:
  - ***Genericidad (parametrización de tipo)***: el significado de un tipo de datos está definido en base a unos parámetros de tipo que pueden variar

Funciones genéricas:

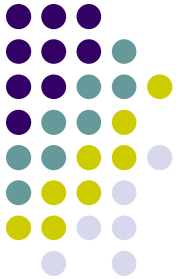
`OrdenaInt(int array[]), OrdenaFloat(float array[]),...`

➔ `Ordena<T> (T array[])`

Clases genéricas:

`PilaInt, PilaChar, PilaFloat, ...` ➔ `Pila<T>`

Introducción a C++



- Funciones y clases genéricas:
  - **C++**: definición mediante sentencias '**`template`**'

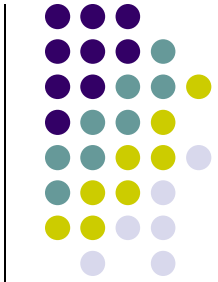
```
template < parámetros genéricos > función o clase genérica
```

    - *Parámetros genéricos*: lista con uno o varios tipos genéricos (clases)

```
template<class A, class B>
template<class T>
```

    - *Función o clase genérica*: declaración normal de una función o una clase pudiendo usar los tipos de la lista de parámetros genéricos.

Introducción a C++

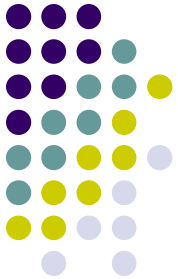


- Funciones y clases genéricas:
  - Funciones genéricas
    - *Definición*:

```
template < parámetros genéricos >
tipoDevuelto  nombreFunción  ( parámetros de la función )
{
   cuerpo de la función
}
```

      - *Utilización*: llamar directamente a la función (el compilador instancia la versión adecuada)
      - ***Ejemplo*: Funciones `ordena` y `escribe` (págs. 2 y 3)**

Introducción a C++



- Funciones y clases genéricas:
  - Clases genéricas
    - *Definición*:

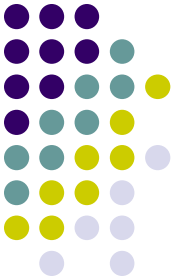
```
template < parámetros genéricos >
class  nombreClase
{
    definición de miembros de la clase
};
```

    - *Implementación métodos*:

```
template < parámetros genéricos >
tipoDevuelto  nombreclase<par>::nombreFunción  ( parámetros )
{
   cuerpo de la función
}
```

    - *Utilización*: instanciar la clase a un tipo concreto
    - ***Ejemplo*: clase genérica pila<T> (págs. 3 y 4)**

Introducción a C++



- Funciones y clases genéricas:

PROBLEMA

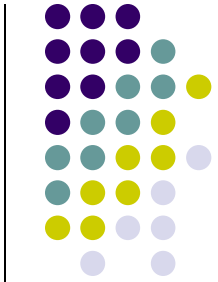
Si separamos la definición de una clase genérica en fichero de cabecera (`clase.hpp`) y de implementación (`clase.cpp`), la compilación (`g++ -c pila.cpp`) no genera código objeto. Si otro módulo usa la clase genérica (`#include clase.hpp`), el compilador dirá que no encuentra la implementación de la clase.

SOLUCIÓN: para las clases genéricas, tanto la definición como la implementación de la clase deben ir en el mismo fichero, el **`hpp`**.

Introducción a C++

- Excepciones:
  - ¿Qué hacer ante un **error irreversible**? (div 0, imposible reservar memoria o leer fichero, …)
    1. Mostrar **mensaje de error**: `cerr << "Error: la pila está vacía";`
    2. Devolver un **valor especial** que signifique error
    3. **Interrumpir la ejecución** del programa: `exit(1)`
    4. No hacer **nada** y continuar
    5. **Provocar** o lanzar una **excepción**.

Introducción a C++



- Excepciones (*significado*):
  1. El procedimiento que **lanza la excepción** (***A***), interrumpe su ejecución en ese punto.
  2. El procedimiento que ha llamado al ***A*** (***B***), debe tener código para **manejar excepciones**.
  3. Si ***B*** no maneja la excepción, pasa al que ha llamado a ***B*** → **propagación de la excepción**
  4. Si la excepción se propaga hasta el `main()` y este no maneja la excepción, se interrumpe la ejecución del programa mostrando **error**.

Introducción a C++

- Excepciones:
  - Definición: conceptualmente, un objeto

```
class DivisionPorCero {};
```

  - Provocar excepción:

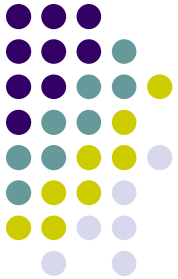
```
throw DivisionPorCero();
```

  - Manejar una excepción:

```
try {
    sentencias1
}
catch (nombreClase_1) {
    sentencias2
}
catch (nombreClase_2) {
    sentencias2
}
catch (...) {
    sentencias2
}
```

  - ***Ejemplo pág. 6***

Introducción a C++



- Asertos:
  - Verificar una condición
    - Si se cumple, seguir normalmente.
    - Si no, se muestra la línea de código donde falló el programa y se interrumpe la ejecución.
  - Definición: `assert(condición booleana)`
  - Necesaria librería `<assert.h>`
  - Pre-/Post-condiciones:

```
funcion (parametros)
{
   assert (precondición);
   ...
   // Cuerpo de la función
   ...
   assert (postcondición);
}
```

Introducción a C++

- Asertos:
  - Ejemplo
  - Problema:
    - ¡Los asertos provocan la interrupción del programa!
    - Posible solución: simular con excepciones

```
double divide (int a, int b)
{
  assert(b!=0);
  return double(a)/double(b);
}
```

Desactivar comprobación :
-DNDEBUG

```
class FalloPrecondicion {};
class FalloPostcondicion {};

void precondicion (bool condicion)
{
#ifndef NDEBUG
    if (!condicion)
        throw FalloPrecondicion();
#endif
}

void postcondicion (bool condicion)
{
#ifndef NDEBUG
    if (!condicion)
        throw FalloPostcondicion();
#endif
}
```

Introducción a C++

- El puntero `this`:
  - `this`: variable especial dentro de cualquier clase T
  - El tipo de `this` es un puntero a T
    - Puntero al objeto receptor del mensaje
  - Ojo: en muchos casos, '`this->XXX`' se puede sustituir por '`XXX`'; el uso de `this` hace el código menos legible.

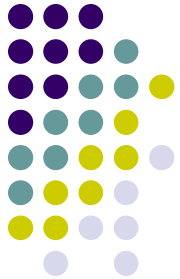
Ejemplo página 8

Introducción a C++

- Redefinición de operadores
  - **Operadores**: funciones básicas del lenguaje (`+`, `-`, `*`, `=`, `==`, `<<`, `>>`, `&&`, `||`, etc.)
  - **Redefinición de operadores**: establecer cómo se debe comportar un operador dentro de una clase
    - Definir e implementar funciones con el nombre:
      - `operator+`
      - `operator-`
      - `operator*`
      - `operator==`
      - …

Ejemplo págs. 10-12

Introducción a la STL

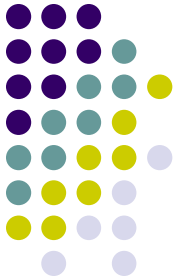


- **STL (Standard Template Library)**: librería estándar de estructuras de datos y algoritmos que forma parte del estándar del C++

- **Objetivo**: evitar que los programadores tengan que programarse una y otra vez los algoritmos y estructuras de datos fundamentales.

Descargar: http://dis.um.es/~frgarcia/files/doc/aed/guiastl.pdf

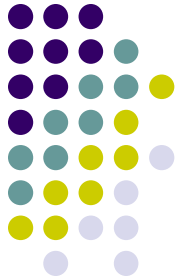
Introducción a la STL



- Listas en la STL de C++: `list<T>`
  - **Lista**: secuencia de elementos donde se puede insertar y eliminar elementos rápidamente, pero que no puede accederse a ellos según un índice.
    - La clase `list` provee una estructura genérica de listas enlazadas pudiendo eventualmente contener repeticiones.
    - Dificultad:
      - Inserción (al principio o fin de lista): `O(1)`
      - Búsqueda: `O(n)` en general, `O(1)` para el primer y el último eslabón

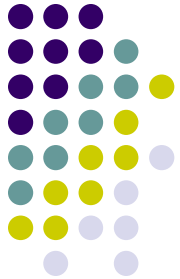
http://www.cplusplus.com/reference/stl/list/

Introducción a la STL



- Listas en la STL de C++: `list<T>`
  - `#include <list>`
  - Operaciones básicas (`list<T> l`):
    - `l.push_back(e)`: inserta un elemento `e` de tipo `T` final
    - `l.push_front(e)`: inserta un elemento `e` de tipo `T` principio
    - `l.front()`: obtener primer elemento de la lista
    - `l.back()`: obtener último elemento de la lista
    - `l.pop_front()`: eliminar el primer elemento de la
    - `l.pop_back()`: eliminar el último elemento de la

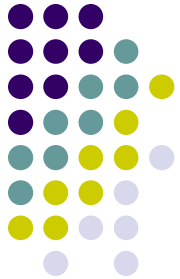
Introducción a la STL



- Listas en la STL de C++: `list<T>`
  - Operaciones básicas (`list<T> l`):
    - `l.empty()`: saber si la lista está vacía
    - `l.size()`: conocer el número de elementos✵
    - `l.clear()`: borrar la lista
    - …

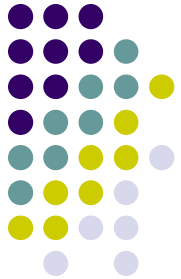
✵ Precisa de tiempo lineal (`O(n)`); ¡utilizar lo menos posible!

Introducción a la STL



- Listas en la STL de C++: `list<T>`
  - Operaciones con iteradores:
    - Para trabajar con elementos interiores de la lista es necesario usar iteradores
    - `l.begin()`: iterador al primer elemento de la lista
    - `l.end()`: iterador que apunta una posición del último
    - Desplazamiento del iterador (it) por la lista: `++it` `--it`
    - Recuperar elemento apuntado por iterador: `*it`
    - `l.erase(it)`: eliminar elemento de la lista
    - `l.insert(it,e)`: insertar antes del elemento `it`

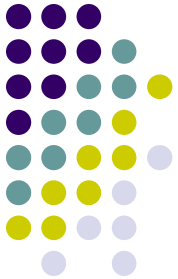
Introducción a la STL



- Listas en la STL de C++: `list<T>`
  - Otras operaciones: algoritmos programados específicamente para listas que pueden ser útiles
    - `l.reverse()`: girar una lista
    - `l.unique()`: eliminar elementos repetidos que aparezcan consecutivamente
    - `l1.merge(l2)`: fusionar `l2` en `l1` obteniendo con todos los elementos ordenados
    - `l.sort()`: ordenar la lista [¡no
    - `l.sort(compara)`: indica funció

```
bool compara(const
T &a, const T &b)
```

Planificación práctica



Semana 3. 5 de noviembre: el intérprete de comandos (004), implementación sencilla del diccionario de productos con arrays (005) y con listas (006).

- 005
  - Intérprete de comandos para reconocer todos los comandos de la aplicación: **insertar**, **palabras**, **precios**, **eliminar** y **producto**

```
string comando;
while (cin >> comando) {
   if (comando=="insertar") procesarInsertar();
   else if (comando=="palabras") procesarPalabras();
   else if (comando=="precios") procesarPrecios();
   ...
}
```

Planificación práctica

Semana 3. 5 de noviembre: el intérprete de comandos (004), implementación sencilla del diccionario de productos con arrays (005) y con listas (006).

- 006
  - Colección de productos: almacenar productos como lista de elementos
    - Operaciones:
      - `insertar`,
      - `eliminar`,
      - `producto`,
      - `precios`

Planificación práctica

Semana 3. 5 de noviembre: el intérprete de comandos (004), implementación sencilla del diccionario de productos con arrays (005) y con listas (006).

Sugerencias de Implementación

Recomendamos utilizar para este ejercicio las listas de las STL (las librerías estándar de C++): el tipo **list<T>**. Para aprender a usar las listas puedes consultar:

- Brevísimo manual de introducción a las listas de las STL.
- Referencia de las STL en la web (sección list).
- Manual de uso de las STL para principiantes de la OIE.

Observar que hay diferentes formas de almacenar los datos en nuestra colección de productos. Por ejemplo, se pueden almacenar en el orden en que llegan, usando la operación **push_front** del tipo **list<T>**; en ese caso, puesto que la operación **precios** requiere que los productos estén ordenados, se podría usar la operación **sort** de las listas. Otra posibilidad sería almacenar los productos en la lista ordenados por identificador; para ello en cada inserción habría que usar un iterador de listas y la operación **insert** de **list<T>**.

Modularidad

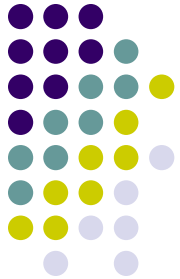
Nuestro programa empieza ya a tener cierto tamaño. Por lo tanto, va siendo necesario aplicar los principios de la **programación modular** para organizar mejor el código. Debemos descomponer el código en diferentes módulos asociados a las distintas funcionalidades que aparecen. Una vez hecha la descomposición modular, ¿cómo enviar el programa al juez?

La solución es usar ficheros TAR. El juez on-line admite como entrada un fichero TAR, que contiene los archivos de código C/C++ necesarios y el `Makefile` para construir el ejecutable. Se deben seguir los siguientes pasos:

- Estando situados dentro del directorio donde se encuentran todos los ficheros de código, ejecutar:
  ```
  >> tar -cf archivo.tar *.cpp *.h Makefile
  ```
- Observar que el parámetro `archivo.tar` indica el nombre del fichero que se quiere generar, y a continuación vienen los archivos a incluir en el TAR. No se deben usar ficheros TAR comprimidos.
- El archivo `Makefile` debe contener las instrucciones adecuadas para compilar el proyecto y generar un **ejecutable llamado** `a.out`. Repetimos: el ejecutable resultante debe llamarse: `a.out`
- El juez on-line: (1) toma el fichero TAR; (2) extrae los archivos `*.cpp`, `*.h`, `*.c`, `*.hpp` y `Makefile`; (3) ejecuta "`>> make`"; y (4) ejecuta "`>> ./a.out`". Si falla alguno de estos pasos, se obtendrá un error.

A partir de este ejercicio y en los sucesivos, se requiere que los programas sean subidos al juez en formato TAR. En los seminarios de repaso de C puedes encontrar una guía abreviada del uso de Makefile.

Planificación práctica



Semana 3. 5 de noviembre: el intérprete de comandos (004), implementación sencilla del diccionario de productos con arrays (005) y con listas (006).

- `007` (opcional)
  - Comando "`palabras`" con listas
    - Buscar todos los productos que contengan todas las palabras pasadas como parámetro (en `nombre` o `descripción`)
    - *Palabra*: como en **ejercicio 001**
    - Búsqueda independiente de mayúsculas/minúsculas: usar función de **ejercicio 003**
    - Resultado: listado de productos que cumplen él criterio ordenados por identificador