PARADIGMAS DE PROGRAMACIÓN
Licenciatura en Informática
Programador Universitario

# UNIDAD VI
# PARADIGMAS DE PROGRAMACIÓN
## Plantillas (templates)
## Contenedores - Iteradores

# Templates

**SOBRECARGA** → **Cualidad polimórfica** de un elemento de poderse aplicar a diferentes tipos de O

O de distinto tipo pueden ejecutar operaciones con el mismo nombre y distinta semántica

**Ejemplo Sobrecarga en C++**

```
void imprimir(int x){
 cout<<"int"<<endl<<x<<endl;
}
void imprimir(char x){
cout<<"char"<<endl<<x<<endl;
}
void imprimir(string x){
 cout<<"string"<<endl<<x<<endl;
}
```

```
main () {
          int i;
          char c;
          string s;
          ... ...
          imprimir(i);
          imprimir(c);
          imprimir(s);
          ... ...
}
```

# Templates

**POLIMORFISMO** → **Calidad o estado de un elemento de asumir varias formas (RAE)**

**POLIMORFISMO PARAMÉTRICO** → **Capacidad de definir tipos genéricos** que dependiendo del tipo del mensaje que actúe sobre la estructura genérica, se genere un tipo específico de estructura

El **elemento se define como una plantilla** que implementa la misma ejecución pero con diferentes tipos de datos

**Ej. Plantilla de función imprimir en C++**

```
template <class ITEM>
void imprimir (ITEM dato){

cout<<typeid(ITEM).name()<<endl;
  cout<<dato<<endl;
}
```

```
main(){
…
imprimir(20);
imprimir('A');
imprimir("String");
…}
```

3

# Templates

**POLIMORFISMO PARAMÉTRICO**

Se aplica satisfactoriamente sólo en **LENGUAJES TIPADOS**

**GENERICIDAD**

Capacidad que tiene un lenguaje de formar **funciones** o **clases paramétricas** que describen estructuras de datos genéricas

**Función Paramétrica**

```
template < class ITEM >
void imprimir (ITEM x){…}
```

**Clase Paramétrica**

```
template < class ITEM >
class Pila{…};
```

**Parámetro Genérico Formal**

**VENTAJA**

- Eliminar sentencias condicionales del tipo SWITCH
- Compartir código

4

# Templates

```
bool Pertenece(int *arre, int MAX, int item ){
    int i=0;
    bool encontrado=false;
    while(!encontrado && i<MAX){
        cout <<endl<< "arreglo["<<i<<"]:"<<arre[i]<<endl;
        if (arre[i]==item)
                encontrado = true;
        i++;
    }
    return encontrado;
}
```

```
bool Pertenece(char *arre, int MAX, char item ){
    int i=0;
    bool encontrado=false;
    while(!encontrado && i<MAX){
        cout <<endl<< "arreglo["<<i<<"]:"<<arre[i]<<endl;
        if (arre[i]==item)
                encontrado = true;
        i++;
    }
    return encontrado;
}
```

5

# Templates

**Plantillas de Función ó Funciones Genéricas** → Proporcionan el mecanismo adecuado para **parametrizar un conjunto de tipos de interfaz de la función** (los argumentos y el tipo de retorno) , mientras que el cuerpo de la misma permanece invariante

```
template <class T>
void imprimir (T x){...}
```

Ver **PlantillaFuncion** en Eclipse

```
main(){
…
imprimir(20);//Fcion. plantilla
imprimir('A');//Fcion. plantilla
imprimir("xxx");//Fcion. plantilla
…}
```

En tiempo de compilación, el tipo paramétrico T **es sustituido** por el tipo actual de la instancia que invoca a la función.

**Función plantilla** (ó **función de plantilla**) → Instanciación específica de una función genérica (o plantilla de función)

6

# Templates

**Plantillas de Clase ó Clase Genérica** → Cada vez que se necesite una nueva clase, se le indica al compilador el tipo y éste escribe el código fuente para la nueva clase

```
template < class ITEM >
class Pila{…};
```

```
main() {
  Pila<int> pInt; //Clase de Plantilla
  Pila<Fecha> pFecha; //Clase de Plantilla
…}
```

Instanciación específica de una Plantilla de Clase

**Clases de Plantillas ó Tipos Parametrizados** → Requieren uno o más **parámetros de tipo** para especificar la manera de personalizar una Plantilla de Clase

Las clases diseñadas para contener colecciones de objetos de un determinado tipo se denominan **contenedores genéricas** y son buenos ejemplos de plantillas de clases

7

Ver **PlantillaPila** en Eclipse

# Templates

**Plantillas de Clases ó Clases Genéricas** → Permiten definir un modelo para definiciones de clases

**Sintaxis**

La palabra **template** encabeza la declaración/definición de la clase genérica, seguida de una lista de parámetros formales de la plantilla encerrada entre **< >**

**Clase Genérica**

```
template <class item>
class Pila{
        int tope;
        item *arreglo[MAX];
        static const int MAX=100;
public:
        Pila();
        Pila(const Pila<item> &);
        bool push(item);
        bool pop();
        item top();
        bool esPilavacia();
        void escribir();
        ~Pila();
};
```

**Parámetro Genérico Formal**

El tipo de elemento a almacenarse en esta **Pila** se menciona sólo genéricamente como **item** a lo largo del encabezado de la clase y de las definiciones de función miembro

```
template < class item >
void Pila<item>:: pop(){
    if( tope >= 0 )
        tope--;
}
```

8

# Templates

**Tipos parametrizados** ó **plantillas**

- Pueden tomar más de un parámetro formal, pero cada uno de los ellos debe estar precedido de la palabra class

```
template <class T, class S, class R>
```

- Pueden contener en su lista de parámetros formales argumentos de tipos predefinidos, cuyos valores deben ser expresiones constantes

```
template <class T, int dim>
```

Ver **PlantillaPila2** en Eclipse

- OBS → Si se define explícitamente una Función/Clase de Plantilla para un determinado tipo, esta definición anula la definición automática de la funcion/clase correspondiente a la Funcion/Clase Genérica

Ver **PlantillaFuncion2** en Eclipse

Ver **PlantillaPila** en Eclipse

9

# Clases Contenedores y Clases Iteradores

**Clases Contenedores** ó de **Colección** → Clases diseñadas para contener colecciones de O

Ver **ListaEnlazada** en Eclipse

- Gestionan el espacio de almacenamiento de sus elementos y proporcionan servicios como inserción, búsqueda, clasificación, prueba de un elemento verificando su membresía dentro de la clase, y otras similares

**Clases Iteradores** → Un **iterador** es un O que retorna el elemento siguiente de una colección

Una clase contenedor puede tener varios O Iteradores operando sobre ella simultáneamente

- Generalmente se escriben como amigos de la Clase Contenedor, a través de los cuales se hace la iteración

**Ventajas**

- Permite manipular eficientemente el contenedor manteniendo oculta su representación interna
- Podemos usar iteradores como "punteros seguros" a la estructura sin exponerla
- Permite escribir algoritmos genéricos (asumiendo una interfaz común)

Ver **ListaEnlazada** en Eclipse

10

# Clases Contenedores y Clases Iteradores

**Standard Template Library** — Biblioteca Estándar de Plantillas de C++

Contiene

- **CONTENEDORES** → Plantillas de las estructuras utilizadas en programación: pilas, filas, conjuntos, vectores, árboles, etc.
- **ITERADORES** → Plantillas de Iteradores: trivial, reverso, bidireccional, de acceso aleatorio, etc.
- **ALGORITMOS** → Colección de funciones especialmente diseñadas para procesar los datos de los contenedores a través de los iteradores: buscar, contar, mover, copiar, ordenar, etc.

Para utilizar un contenedor de la STL, el tipo de elemento que está almacenado en el contenedor deberá proporcionar la sobrecarga de los operadores == y <

Ver **ContenedoresIteradores** en Eclipse

11