Templates

Pregunta ¿¿¿???

Venimos programando **TADs** en C y **Clases** en C++.

Pero, si quiero usar una cola de enteros, y luego una cola de números complejos, aunque el comportamiento del Tipo o Clase sea el mismo (COLA), tenemos que implementar repetidamente el tipo o la clase, una por cada tipo de dato que necesite almacenar en la COLA.

Esto para el año 2022… no es muy poco práctico??

Dale, entonces aprendamos a programar TEMPLATES!!

Que tal si programamos una **PLANTILLA (TEMPLATE),** que tenga el **comportamiento** de una cola y luego la **instancio** con **distintos tipos**? Cola de enteros, cola de caracteres, cola de estrellas, etc, etc.

Templates

Un **template o plantilla** son funciones y clases que no están implementadas para un tipo determinado, sino para un tipo que se debe definir en cada momento.

Un **template** es un patrón para **creación de clases o funciones como instancias** de una plantilla en tiempo de ejecución, de igual forma que una **clase es un patrón** para crear **objetos** como instancias de la clase en tiempo de ejecución.

Las plantillas pueden diseñar pilas, colas, conjuntos, etc, genéricos, sin importar el tipo de elemento que procesan.

```
void intercambio(char &var1, char &var2)
{
   ....
}
```

```
void intercambio(tipo_dato var1, tipo_dato var2)
{
 ...
}
```

Templates de clases

Son un artificio C++ que permite definir una clase mediante uno o varios parámetros. Este mecanismo es capaz de generar infinitas clases distintas pero compartiendo un diseño común.

```
class Punto
{
  private:
     int coorx ;
     int coory ;
  public:
     ...
} ;
```

Podemos diseñar la clase Punto con atributos de tipo **int**

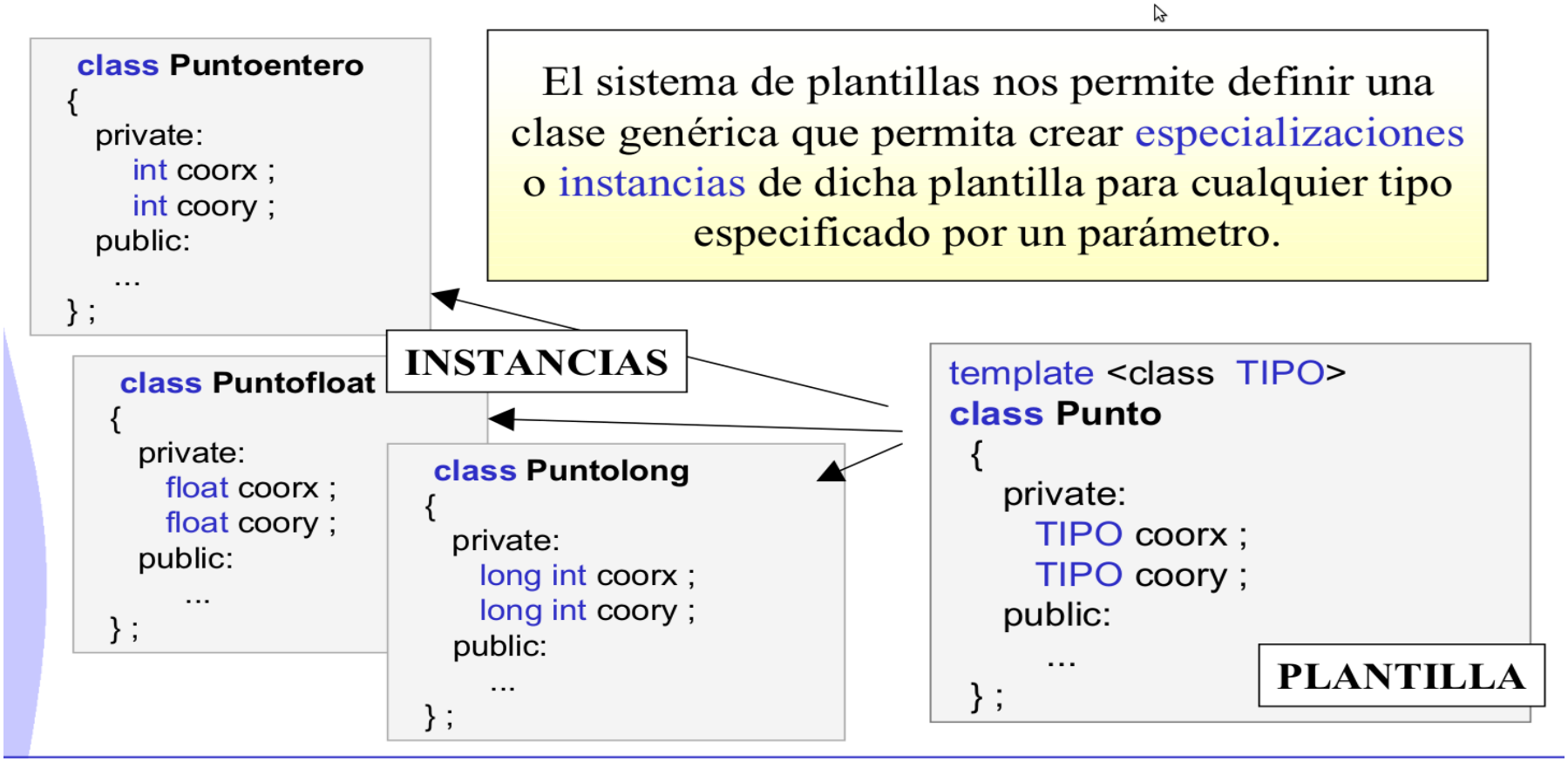
En el caso de que necesitemos que el tipo de los puntos sea de otro tipo, tendremos que crear otra clase distinta e implementar todos los métodos.

Pérdida de tiempo

Posibilidad de error

Templates de clases

Son un artificio C++ que permite definir una clase mediante uno o varios parámetros. Este mecanismo es capaz de generar infinitas clases distintas pero compartiendo un diseño común.



Templates de clases

```
template <class  TIPO>
class Punto
 {
    private:
       TIPO coorx ;
       TIPO coory ;
    public:
        ...
 } ;
```

PLANTILLA

```
 class Punto
{
   private:
      float coorx ;
      float coory ;
   public:
       ...
} ;
```

INSTANCIA

```
 ...
Punto <float> obj ;
...
```

Se escribe el nombre de la plantilla de clase seguido por los tipos con los que se declara entre < >

Esta instrucción genera una clase normal a partir de la plantilla de clase

Instanciación implícita

Template de clases

```
// interfaz de una plantilla de clase para definir pilas
template <class T>
class Pila
{
   T datos[50];
   int elementos;
   public:

   Pila() {}

   void Push(T elem);
   T Pop();
   int Size();
   int Vacia();

}

// utilizacion
Pila <int> pila_int;
Pila <float> pila_real;
```

Template de clases

```
// interfaz de una plantilla de clase para definir pilas
template <class T>
class Pila
{
   T datos[50];
   int elementos;
   public:

   Pila() {}

   void Push(T elem);
   T Pop();
   int Size();
   int Vacia();

}

// utilizacion
Pila <int> pila_int;
Pila <float> pila_real;
```

Definición de plantilla: comienza con

la palabra reservada “template”, seguida por una lista de parámetros de la plantilla.

Parametros de tipo entre < >

Templates de funciones

Las funciones genéricas pueden ser miembros (métodos) de clases.

```
class MyClase
{
   public:
      template <class  Tipo>
      Tipo maximo ( Tipo a, int b );
   private:
       ...
       ...
};
```

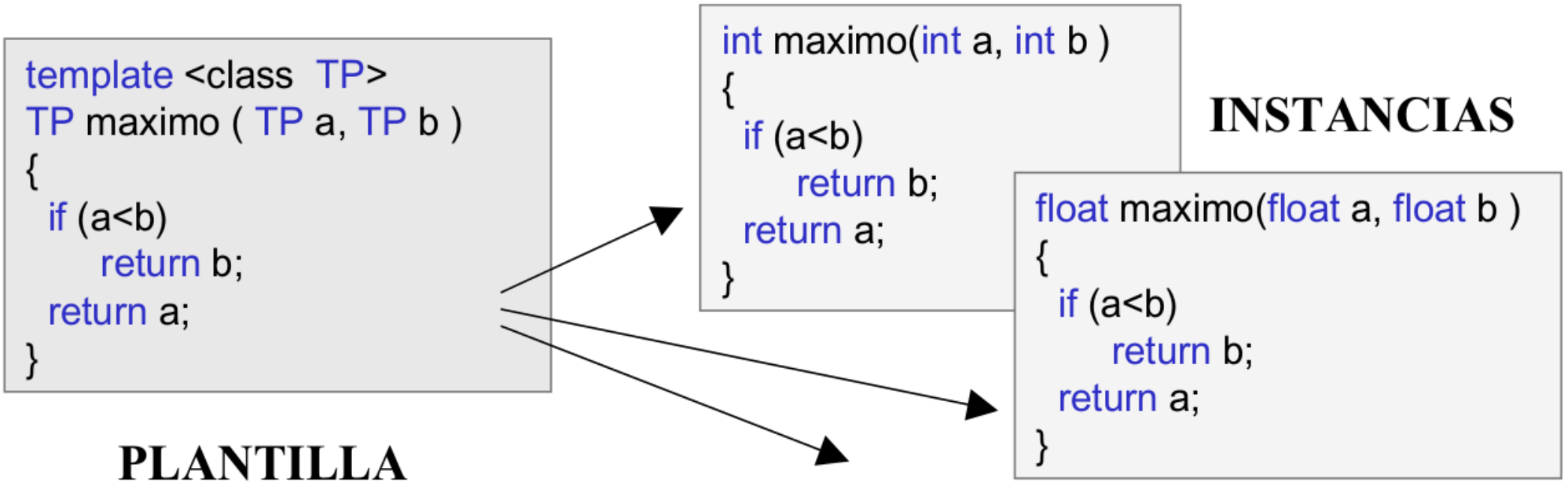
Prototipo

```
template <class  Tipo>
Tipo Myclase::maximo ( Tipo a, Tipo b )
{
  if (a<b)
       return b;
  return a;
}
```

Implementación

Templates de funciones

A partir de plantillas de función el compilador es capaz de generar código de funciones distintas que comparten ciertas características.
Las funciones así generadas se denominan instancias o especializaciones de la plantilla o template.



Templates de funciones

Ejemplo:

```
class ClaseA
{
   private:
         int x ;
   public:
      ClaseA (int a);

      template <class  Tipo>
      void func ( Tipo a, Tipo b );
};
```

```
ClaseA :: ClaseA ( int a )
{
  x = a;
}
```

```
template <class  Tipo>
void  ClaseA::func ( Tipo a, Tipo b )
{
  cout  << a ;
  cout  << b ;
  cout  << x ;
}
```

```
void main( )
{
 ClaseA obj1(4), obj2(5);
 obj1.func ( 'm' , 'k' );
 obj2.func ( 100, 200 );
}
```

Salida

```
m , k , 4
100, 200, 5
```