

# *Programming Primer II*

## Templates, Iteración y Recursión

Programación de Estructuras de Datos y Algoritmos Fundamentales  
(TC1031)

M.C. Xavier Sánchez Díaz  
sax@tec.mx



# Outline

- 1 Templates
- 2 Patrones de diseño de algoritmos

# Data Types en C++

## Templates

C++ es un lenguaje *fuertemente tipado*—funciones, objetos y estructuras deben tener un tipo de dato asociado al compilar y es el único tipo de datos que aceptarán.

### Boolean return with integer parameters

```
1 bool greaterThanInt(int a, int b){  
2     bool g = false;  
3     if (a > b){  
4         g = true;  
5     }  
6     return g;  
7 }
```

# Data Types en C++

## Templates

### Boolean return with double precision floating-point params

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <math.h>
4
5  bool approxEqual(double x, double y, double threshold=0.00005){
6      bool g = false;
7      if (fabs(x - y) <= threshold){
8          g = true;
9      }
10     return g;
11 }
12
13 int main(int argc, char* argv[]){
14     double x = atof(argv[1]);
15     double y = atof(argv[2]);
16
17     std::cout << approxEqual(x, y);
18     return 0;
19 }
```

# Templates en C++ I

## Templates

Podemos convertir una *función tipada* en un *template* para poder usar múltiples tipos de datos al incluir el decorador **template** <class mytype> justo arriba de una **clase** o **función** para hacerla *genérica*:

### Defining a template

```
Lista.h
1  #define MAX 100
2  template <class T>
3  class Lista
4  {
5  private:
6      T data[MAX];
7      int size;
8  public:
9      Lista();
10 ...
```

# Templates en C++ II

## Templates

Después, durante la implementación, especificamos el tipo de dato específico a utilizar, por ejemplo:

### Implementing a template

miLista.cpp

```
1 #include "Lista.h"
2 int main(){
3     Lista<float> miLista;
4     ...
```

para implementar una lista que guarde solamente números decimales.

# Patrones de diseño

## Iteración

En el ámbito de software, llamamos **patrones de diseño** a la manera en la que se 'resuelve' un problema. Existen muchos patrones de diseño para algoritmos, pero en el curso revisaremos dos enfoques importantes:

- **Métodos Iterativos.** Se usan para recorrer contenedores con estructuras indizables.
- **Métodos Recursivos.** Se usan para recorrer o generar contenedores con estructuras recursivas

Cada uno tiene sus propiedades y tienen fundamentos matemáticos distintos.

# Patrones de diseño

## Iteración

En el ámbito de software, llamamos **patrones de diseño** a la manera en la que se 'resuelve' un problema. Existen muchos patrones de diseño para algoritmos, pero en el curso revisaremos dos enfoques importantes:

- **Métodos Iterativos.** Se usan para recorrer contenedores con estructuras indizables.
- **Métodos Recursivos.** Se usan para recorrer o generar contenedores con estructuras recursivas

Cada uno tiene sus propiedades y tienen fundamentos matemáticos distintos.



# Patrones de diseño

## Iteración

En el ámbito de software, llamamos **patrones de diseño** a la manera en la que se 'resuelve' un problema. Existen muchos patrones de diseño para algoritmos, pero en el curso revisaremos dos enfoques importantes:

- **Métodos Iterativos.** Se usan para recorrer contenedores con estructuras indizables.
- **Métodos Recursivos.** Se usan para recorrer o generar contenedores con estructuras recursivas

Cada uno tiene sus propiedades y tienen fundamentos matemáticos distintos.

# Patrones de diseño

## Iteración

En el ámbito de software, llamamos **patrones de diseño** a la manera en la que se 'resuelve' un problema. Existen muchos patrones de diseño para algoritmos, pero en el curso revisaremos dos enfoques importantes:

- **Métodos Iterativos.** Se usan para recorrer contenedores con estructuras indizables.
- **Métodos Recursivos.** Se usan para recorrer o generar contenedores con estructuras recursivas

Cada uno tiene sus propiedades y tienen fundamentos matemáticos distintos.

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$



# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ \textcolor{red}{32} & 64 \end{bmatrix}$$

# Iteración

## Patrones de diseño de algoritmos

La **iteración** está basada en las **estructuras matemáticas ordenadas** como las tuplas, las cadenas de caracteres y los arreglos  $n$ -dimensionales.

$$\langle 1, 2, 3, \dots \rangle$$

$$\begin{bmatrix} 2 & 4 \\ 8 & 16 \\ 32 & 64 \end{bmatrix}$$

# Recursión

## Patrones de diseño de algoritmos

La **recursión** es una propiedad estructural de algunos objetos matemáticos (como los números naturales  $\mathbb{N}$ ) que se puede simplificar en dos pasos:

- 1 Existe un caso base
- 2 Existe una regla general basada en el caso anterior para todos los siguientes casos

### Inducción en $\mathbb{N}$

- Caso base: 1
- Regla general: sumar 1 al número anterior

Con esto podemos generar **todos** los números naturales  $(1, 2, 3, 4, \dots)$ :

$$1, (1) + 1, ((1) + 1) + 1, (((1) + 1) + 1) + 1, \dots$$

# Recursión

## Patrones de diseño de algoritmos

La **recursión** es una propiedad estructural de algunos objetos matemáticos (como los números naturales  $\mathbb{N}$ ) que se puede simplificar en dos pasos:

- 1 Existe un caso base
- 2 Existe una regla general basada en el caso anterior para todos los siguientes casos

### Inducción en $\mathbb{N}$

- Caso base: 1
- Regla general: sumar 1 al número anterior

Con esto podemos generar **todos** los números naturales  $(1, 2, 3, 4, \dots)$ :

$$1, (1) + 1, ((1) + 1) + 1, (((1) + 1) + 1) + 1, \dots$$



# Recursión

## Patrones de diseño de algoritmos

La **recursión** es una propiedad estructural de algunos objetos matemáticos (como los números naturales  $\mathbb{N}$ ) que se puede simplificar en dos pasos:

- 1 Existe un caso base
- 2 Existe una regla general basada en el caso anterior para todos los siguientes casos

### Inducción en $\mathbb{N}$

- Caso base: 1
- Regla general: sumar 1 al número anterior

Con esto podemos generar **todos** los números naturales  $(1, 2, 3, 4, \dots)$ :

$$1, (1) + 1, ((1) + 1) + 1, (((1) + 1) + 1) + 1, \dots$$

# Recursión

## Patrones de diseño de algoritmos

La **recursión** es una propiedad estructural de algunos objetos matemáticos (como los números naturales  $\mathbb{N}$ ) que se puede simplificar en dos pasos:

- 1 Existe un caso base
- 2 Existe una regla general basada en el caso anterior para todos los siguientes casos

### Inducción en $\mathbb{N}$

- **Caso base:** 1
- **Regla general:** sumar 1 al número anterior

Con esto podemos generar **todos** los números naturales  $(1, 2, 3, 4, \dots)$ :

$$1, (1) + 1, ((1) + 1) + 1, (((1) + 1) + 1) + 1, \dots$$

# Recursión

## Patrones de diseño de algoritmos

La **recursión** es una propiedad estructural de algunos objetos matemáticos (como los números naturales  $\mathbb{N}$ ) que se puede simplificar en dos pasos:

- 1 Existe un caso base
- 2 Existe una regla general basada en el caso anterior para todos los siguientes casos

### Inducción en $\mathbb{N}$

- **Caso base:** 1
- **Regla general:** sumar 1 al número anterior

Con esto podemos generar **todos** los números naturales  $(1, 2, 3, 4, \dots)$ :

$$1, (1) + 1, ((1) + 1) + 1, (((1) + 1) + 1) + 1, \dots$$

# Recursión

## Patrones de diseño de algoritmos

La **recursión** es una propiedad estructural de algunos objetos matemáticos (como los números naturales  $\mathbb{N}$ ) que se puede simplificar en dos pasos:

- 1 Existe un caso base
- 2 Existe una regla general basada en el caso anterior para todos los siguientes casos

### Inducción en $\mathbb{N}$

- **Caso base:** 1
- **Regla general:** sumar 1 al número anterior

Con esto podemos generar **todos** los números naturales  $(1, 2, 3, 4, \dots)$ :

$$1, (1) + 1, ((1) + 1) + 1, (((1) + 1) + 1) + 1, \dots$$

# Factorial

## Iteración

```
1 // iterative factorial
2
3 #include <cstdlib>
4 #include <iostream>
5
6 int factorialIter (int n){
7     int r = 1;
8     for(int i=1; i<=n; i++){
9         r *= i;
10    }
11    return r;
12 }
13
14 int main(int argc, char* argv[]){
15     int n = atoi(argv[1]);
16     std::cout << factorialIter(n);
17     return 0;
18 }
```

# Factorial

## Recursión

```
1 // recursive factorial
2
3 #include <cstdlib>
4 #include <iostream>
5
6 int factorialRecur (int n){
7     // using two return values for the sake of clarity
8     if (n == 1){
9         return 1;
10    }
11    else
12    {
13        return n * factorialRecur(n - 1);
14    }
15 }
16
17 int main(int argc, char* argv[]){
18     int n = atoi(argv[1]);
19     std::cout << factorialRecur(n);
20     return 0;
21 }
```