

# Maestría en Matemática Herramientas Computacionales

Jose Alvarenga

UNAH

PACIII2023

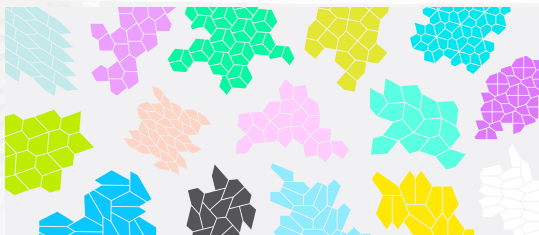
# Tabla de Contenidos

- 1 Motivación
- 2 Programación Estructurada y Orientada a Objetos (C++)

# Introducción I

La computación matemática en la investigación:

- 1 Resolver problemas de la ciencia mediante la simulación por ordenador. Un ejemplo de interesante es el de *Problemas de Mosaicos con copias de polígonos convexos*. En 2017 Rao (Escuela Normal Superior de Lyon) resolvió este problema.



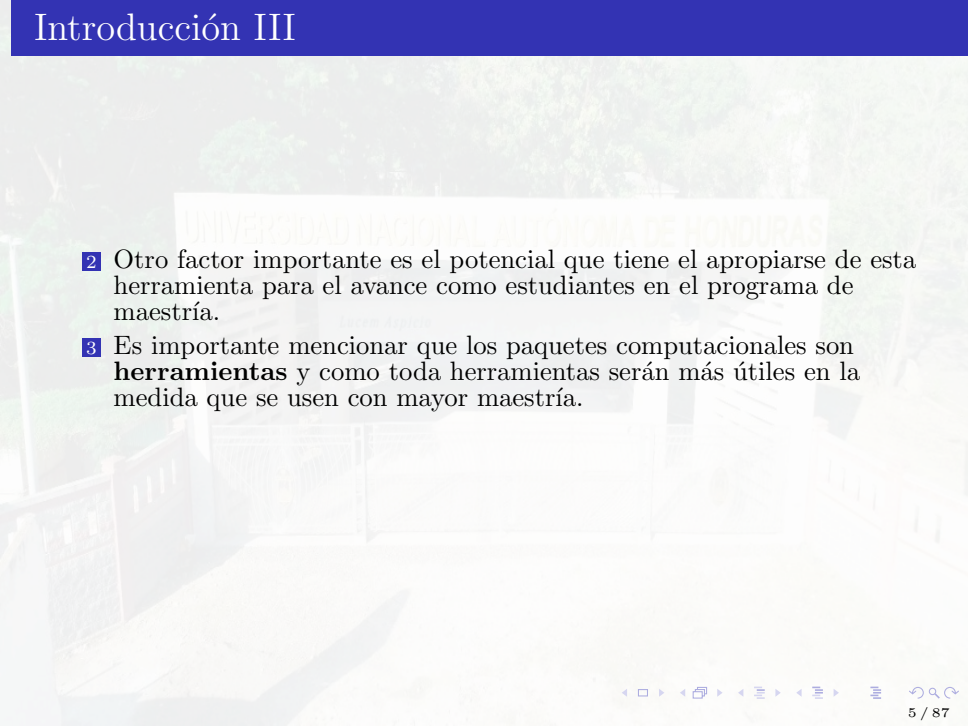
Otras aplicaciones interesantes pueden ser consideradas para estudiar el comportamiento de dinámica de sistemas. Un ejemplo interesante es el estudio del tráfico; en el siguiente enlace podemos ver la dinámica vehicular en una sección de de carretera de varios carriles [\*SimulacionTráfico\*](#).

# Introducción II

- 2 Paquetes estadísticos. Uno de los métodos más famosos usados en paqueterías estadística son los denominados como "Montecarlo". Su origen tiene lugar en el desarrollo de la bomba atómica en los Álamos. Una pequeña descripción del origen es compartida por Nicholas Metropolis:
  - Desarrollo de la computadora ENIAC.
  - Simulación de una reacción termonuclear.
  - Ulam tuvo la idea de aplicar las técnicas de muestreo en conjunto con el nuevo descubrimiento.
- 3 Cálculo simbólico. Realización de cálculos y operaciones simbólicas. Un ejemplo, ¿Cuál es el orden del grupo de Galois de  $p(x) = x^5 - x + 1$ ? El resultado es 120; Maple te permite encontrar este valor con el comando *GroupOrder*.
- 4 Áreas: Criptografía, Optimización, Ecuaciones diferenciales, Topología, etc.

## Observaciones

- 1 Parte de la importancia del curso recae en que el componente computacional es prácticamente inseparable del proceso de formación en la maestría.

- 
- 2 Otro factor importante es el potencial que tiene el apropiarse de esta herramienta para el avance como estudiantes en el programa de maestría.
  - 3 Es importante mencionar que los paquetes computacionales son **herramientas** y como toda herramientas serán más útiles en la medida que se usen con mayor maestría.

# Instalación de Visual Studio y Compilador

Para instalar C++, es necesario contar con un IDE (entorno de desarrollo integrado). En resumen es necesario seguir los siguientes pasos:

- 1 Descargar la versión actual de visual studio code. Puedes encontrarla en la siguiente página [VisualStudioCode](#).
- 2 Instala el compilador de C++. Si tienes un sistema operativo windows, entonces debes instalar el compilador de C++. Puedes encontrarlo en la siguiente página [Msys2](#).
- 3 Si tienes otro sistema operativo, normalmente este ya viene acompañado por un compilador de C++.

Para más detalles consulta el siguiente enlace, de donde se hizo el resumen anterior [VideoTutorial](#).

Para crear un repositorio y actualizarlo en GitHub siga los siguientes pasos:

- 1 Crear una carpeta que se vaya a compartir con la plataforma de GitHub.
- 2 Crear un archivo README.md; dentro de este archivo puede especificar la finalidad del repositorio que se esta creando.
- 3 Dentro de Visual Studio abra la carpeta creada.
- 4 En la opción del *Control de código fuente* se habilitará la opción *Publicar en GitHub*. Sigue los pasos que allí se mencionan.
- 5 Para hacer una actualización de cualquier archivo, ejecute las siguientes instrucciones en una terminal:

```
git status
git add .
git commit -m "Mensaje personalizado"
git push origin main
```



Una motivación del uso del lenguaje C++ se da cuando se desea construir aplicaciones grandes y complejas. Otro aspecto notable es que conceptualmente la dinámica del lenguaje es común a otros lenguajes. Los elementos comunes a la generación de un programa en C++ son los siguientes:

- 1 Código fuente: El texto que representa el programa escrito en un lenguaje de alto nivel.
- 2 Archivo fuente: El archivo que contiene al código fuente.
- 3 Editor: Programa informático para editar el código fuente. Block de notas por ejemplo.
- 4 Sufijo: Es normal que el sufijo del archivo fuente sea cpp.
- 5 Archivo ejecutable: Es el archivo que se genera a partir del código fuente, una vez que este ha sido traducido al lenguaje de máquina. Este archivo normalmente se generará con el sufijo exe con el mismo nombre del archivo fuente.



- 6 Directiva de preprocesador: Son ciertas líneas de código que inicialmente se colocan al inicio del código fuente y tienen el prefijo `#`. Normalmente se usa la directiva `"include"`. A esta directiva se le agrega el nombre de lo que se denomina un archivo de cabecera, el cual contiene un conjunto de rutinas útiles para el desarrollo de nuestro programa.
- 7 Espacio de nombres: Es un contenedor o entorno abstracto creada para contener un conjunto de bloques. Un espacio de nombres comunes es `"std"`, el cual representa a los elementos de la infraestructura estándar de C++. Tiene una función de organización.
- 8 Bloque principal: Esta parte del programa es donde ubicaremos el código que resolverá nuestro problema.

# Tipo de datos primitivos y operadores I

En el aspecto numérico existen dos tipos de representaciones: Los enteros y los de coma flotante (floating point). A continuación se detallan algunas características de este tipo de datos:

- Tipo: Es una característica del número, normalmente mide la representación y el tipo de rango.
- Apóstrofe: Una forma natural de escribir enteros en C++ para un mejor lectura, por ejemplo, en lugar de 1111 se puede escribir 1'111.
- Si se precede un número con cero, se entenderá que este tiene una representación octal.
- Si se precede un número con cero y x (0X o 0x), se entenderá que este tiene una representación hexadecimal.
- Si se precede un número con cero y b (0B o 0b), se entenderá que este tiene una representación binaria.
- Notación científica: xEy, donde x es una expresión decimal y y es un entero; en este caso xEy representa  $x \times 10^y$ .

En C++ también podemos encontrar datos y operadores relacionados con la lógica de proposiciones:

- Los datos de tipo lógico representan a dos tipos de valores, verdadero (true) o falso (false). En C++ a estos tipos se les declara como *bool*.

# Tipo de datos primitivos y operadores II

- Conjunción  $\&$ . Disyunción  $\|$ . Negación  $!$ . Comparación  $==$ .  
Operadores de orden  $>$ ,  $<$ ,  $>=$ ,  $<=$ . El operador de comparación para la diferencia  $!=$ .

En la siguiente tabla se muestra el funcionamiento de algunos operadores lógicos.

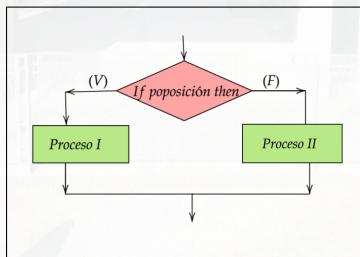
$p$	$q$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$\neg p$
V	V	V	V	V	F
V	F	F	V	F	F
F	V	F	V	V	V
F	F	F	F	V	V

**Figura:** En la tabla se puede apreciar el comportamiento de los operadores lógicos de conjunción, disyunción, implicación y negación respectivamente.

## Paradigma de la programación estructurada

Según este paradigma, se establece que en cualquier algoritmo se pueden identificar los siguientes bloques de repetición:

- Bloques secuenciales.
  - Bloques de selección (if, if-else, switch).
  - Bloques de repetición (while, for, do-while, continue, break).
- El bloque estándar de selección es el denominado *bloque If-Else*.



- Es posible anidar los bloques If-Else de las siguientes formas:

# Programación estructurada II

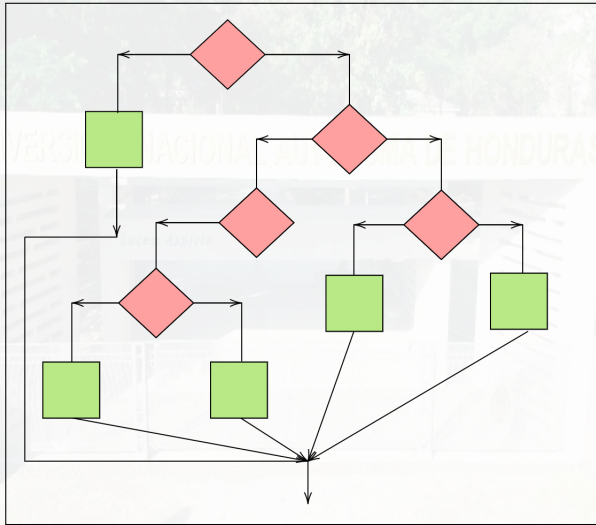
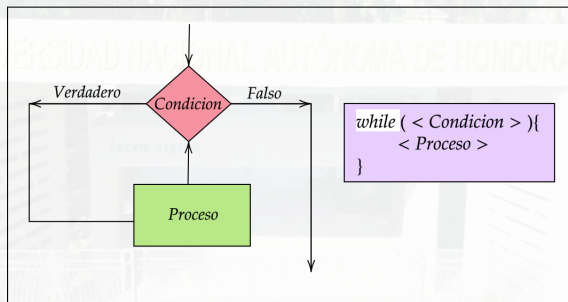


Figura: En el diagrama se observa el estilo anidado, del bloque If-Else.

# Programación estructurada III

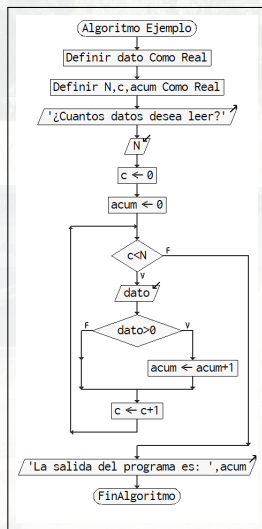
- El bloque de repetición más estándar es el bloque while. El funcionamiento del bloque while se muestra a continuación:



**Figura:** En el diagrama se observa el bloque mientras.

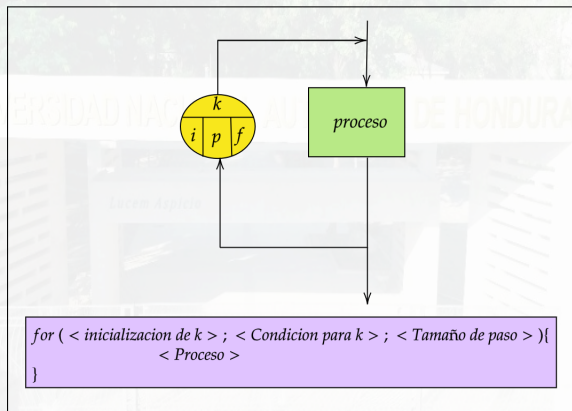
**Ejercicio:** Escriba el código en C++ correspondiente al siguiente diagrama de flujo.

# Programación estructurada IV



- El bloque de repetición *for*, es una estructura de control de repetición que se manipula a través de una variable contadora.



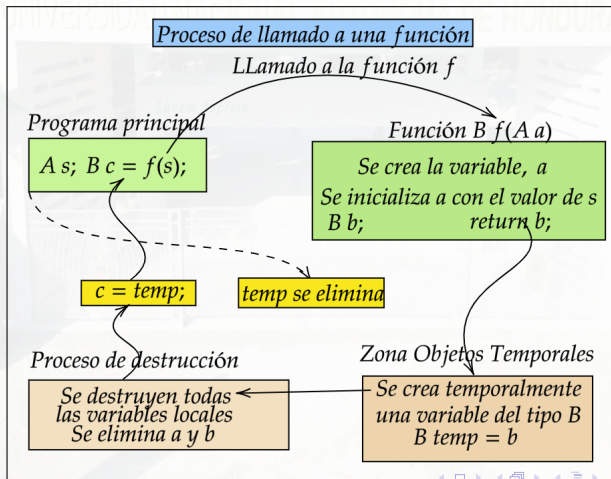


**Figura:** En la figura se observa el código y el diagrama de flujo de un bloque `for`.

# Funciones I

Los elementos más importantes alrededor de una función son:

- 1 Prototipo.
- 2 Implementación.
- 3 Invocación.



Dentro del prototipo de las funciones existen dos partes notables: *el retorno de la función y los parámetros de la función*. Como una característica importante, se sabe que tanto el retorno como los parámetros se pueden mandar o recibir por referencia. El siguiente ejemplo dejará claro cuál es la diferencia alrededor de estos aspectos:

- Al invocar a  $f4$  se crea una copia del valor de  $x$ . Además internamente se crea una variable "sin nombre" para dar el retorno de la función.
- Al invocar a  $f3$  se hace un paso por referencia. Esto implica que no se creará una copia del argumento, en su lugar se seguirá trabajando con la variable que se pase en los parámetros.
- Al invocar a  $f2$  habrá un problema; al hacer un retorno por referencia en el momento que esta función termine, la referencia se perderá y por lo tanto no existirá manera de retornar ese valor.
- En el caso de  $f1$  no existirá el mismo problema dado que la referencia que se envía, continúa existiendo dentro del programa principal.

Código 1: Ejemplo de paso y retorno por referencia

```
1 #include <iostream>
2 using namespace std;
3 int & f1(int &x){
4     x+=1;return x;}
5 int & f2(int x){
6     x+=1;return x;}
7 int  f3(int &x){
8     x+=1;return x;}
9 int  f4(int x){
10    x+=1;return x;}
11 int main(){
12     int x=10;int y;
13     y=f4(x);
14     y=f3(x);
15     y=f2(x);
16     y=f1(x);
17     return 0;}
```

## Código 2: Ejemplo de una función creada por el programador

```
1 #include <iostream>
2 using namespace std;
3 int Div(int a, int b){
4     return a/b;
5 }
6 int main(){
7     int x;
8     x=10;
9     cout<<" Invocacion : _"<<Div(x,20)<<endl;
10    return 0;}
```

## Código 3: Cambiando el orden de la codificación

```
1 #include <iostream>
2 using namespace std;
3 //Definicion de prototipo
4 int Div(int , int);
5 int main(){
6     int x;
7     x=10;
8     cout<<" Invocacion : _"<<Div(x,20)<<endl;
9     return 0;}
10 int Div(int x, int y){
11     return x/y;
12 }
```



Stroustrup, Bjarne (2013). *El lenguaje de programación C++ Cuarta edición.*