

Introducción a la Computación (para Matemáticas)

Primer Cuatrimestre de 2018



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Algoritmo

Un **algoritmo** es una secuencia finita de **instrucciones**.

Ejemplo:

Ingredientes: 15 huevos, 600 gramos de harina, 600 gramos de azúcar

- 1.- Mientras no estén espumosos, batir los huevos junto con el azúcar,
- 2.- agregar la harina en forma envolvente sin batir,
- 3.- batir suavemente,
- 4.- colocar en el horno a 180 grados,
- 5.- si le clavo un cuchillo y sale húmedo, entonces ir a 4.-
- 6.- retirar del horno,
- 7.- mientras no esté frío, esperar
- 8.- desmoldar y servir

Instrucción

Una **instrucción** es una operación que:

- transforma los datos (el *estado*), o bien
 - modifica el flujo de ejecución.
-
- 1.- Mientras no estén espumosos, batir los huevos junto con el azúcar,
 - 2.- agregar la harina en forma envolvente sin batir,
 - 3.- batir suavemente,
 - 4.- colocar en el horno a 180 grados,
 - 5.- si le clavo un cuchillo y sale húmedo, entonces ir a 4.-
 - 6.- retirar del horno,
 - 7.- mientras no esté frío, esperar
 - 8.- desmoldar y servir

Programa

Un **programa** es una secuencia finita de **sentencias**.

Instrucciones



Sentencias

Repaso Taller: presentamos el lenguaje C/C++

- C y C++ son lenguajes compilados: Los archivos .c/.cpp con el código fuente son traducidos a lenguaje de máquina (en archivos .exe), que es ejecutable por el procesador.
 1. El lenguaje de máquina depende de la plataforma (hardware y sistema operativo).
 2. Se requiere un compilador para la plataforma en cuestión.
 3. El código fuente es el mismo, pero el resultado de la compilación es distinto para cada plataforma.
- Un programa en C es una colección de **funciones**, con una función principal llamada main().

Memoria

Durante la ejecución de un programa, sus datos se almacenan en la **memoria**.

La memoria de una computadora es una secuencia numerada de **celdas** o **posiciones**, en las cuales podemos almacenar datos.

Unidad elemental: el **bit**, que toma valores **0** ó **1**.

8 bits = 1 **byte** → Unidad mínima más usada.

1024 bytes = 1 KB (kilobyte)

1024 KB = 1 MB (megabyte)

1024 MB = 1 GB (gigabyte)

1024 GB = 1 TB (terabyte)

1024 TB = 1 PB (petabyte)

...

Soporte físico: electrónico, magnético, óptico, ...

Memoria

[illegible]

Variable

Una **variable** es un **nombre** que denota la **dirección** de una celda en la memoria, en la cual se almacena un **valor**.

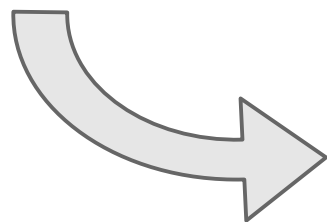
En esa celda de memoria es posible:

- **leer** el valor almacenado, y
- **escribir** un valor nuevo, que reemplace al anterior.

En C++, cada variable tiene asociado un **tipo** (bool, int, float, char, etc.), por lo cual es necesario **declararlas** antes de usarlas.

Ejemplo en C++:

```
int x;    // Declaro la variable x de tipo int.  
x = 10;   // Asigno el valor 10 a la variable x.  
cout << x; // Imprimo en pantalla el valor de x.
```



Para imprimir en pantalla en C++, incluir al principio:
#include <iostream>
using namespace std;

Declaración y asignación de variables

- Todas las variables se deben declarar antes de su uso.
 1. Declaración de la existencia de la variable, con su tipo de datos.
 2. Asignación de un valor a la variable, que no cambia a menos que sea explícitamente modificado por otra asignación.
 3. Inicialización: La primera asignación a una variable.
- Entre la declaración y la inicialización la variable contiene “basura”.

```
int main()  
{ int a = 5; // Declaracion + Inicializacion  
  a = 7+2; // Asignación  
  .....;  
}
```

Tipos de datos

Los programas manipulan **valores** de las **variables** que son de diferentes **tipos**.


Ejemplos:

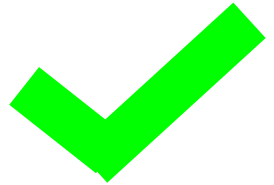
- 1 es un valor de tipo **entero**.
- 2.5 es un valor de tipo **real**.
- “Hola” es un valor de tipo **string**.
- false es un valor de tipo **bool (lógico)**.

Tipos de datos

Enteros (**int**):

Los enteros para una computadora son parecidos a los enteros matemáticos, con una *pequeña* diferencia: están acotados por encima y por debajo.

$-\infty, \dots, -2, -1, 0, 1, 2, \dots, \infty$ 

$-2.147.483.648, \dots, -2, -1, 0, 1, 2, \dots, 2.147.483.647$ 

¿Por qué esas cotas?

Porque lenguajes como C++ o Python usan una **cantidad finita de bits** para representar enteros. Por ejemplo, 32 bits.

Tipos de datos

Operaciones de enteros:

Operador C++	Operación	Ejemplo
+	Suma	$3 + 4 \rightarrow 7$
-	Resta	$6 - 2 \rightarrow 4$
*	Producto	$2 * 8 \rightarrow 16$
/	División	$5 / 2 \rightarrow 2$
%	Resto	$5 \% 2 \rightarrow 1$
-	Negación (unaria)	-6

Tipos de datos

Comparaciones entre enteros:

Operador C++	Operación
<code>i==k</code>	Igualdad
<code>i!=k</code>	Distinto
<code>i<k</code>	Comparación por menor
<code>i>k</code>	Comparación por mayor
<code>i<=k</code>	Comparación por menor o igual
<code>i>=k</code>	Comparación por mayor o igual

Tipos de datos

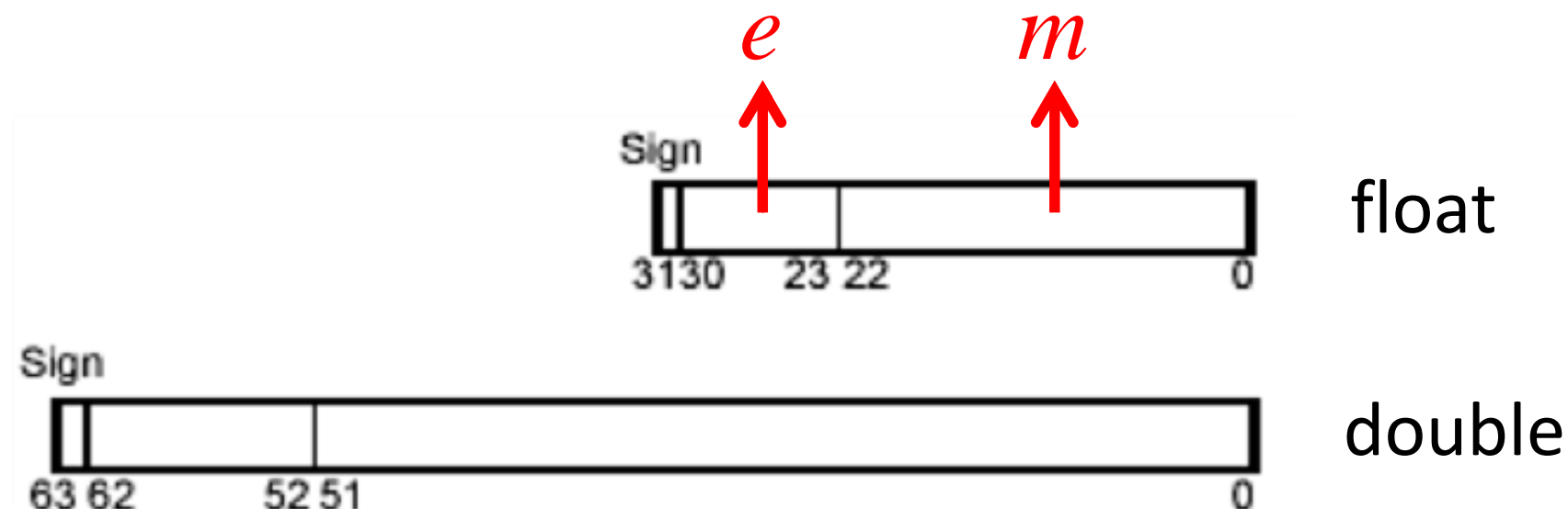
Reales (**float** y **double** en C++):

Un real f representado en punto flotante es un par (m, e) tal que:

$$f \approx \pm m * 10^e \quad \text{donde } 0,1 \leq m < 1$$

(m : mantisa; e : exponente)

Son *bastante* diferentes de los reales matemáticos. Están **acotados por encima y por debajo**, pero también están acotados en la **precisión**.



Tipos de datos

Operaciones de reales:

Operador C++	Operación
+	Suma
−	Resta
*	Producto
/	División
−	Negación (unaria)

Operador C++	Operación
$i==k$	Igualdad
$i!=k$	Distinto
$i<k$	Menor que
$i>k$	Mayor que
$i\leq k$	Menor o igual que
$i\geq k$	Mayor o igual que

(*) No conviene usar $i==k$ entre reales, por los errores de representación. Es probable que querramos que 0.6666667 y 0.6666666 sean considerados *iguales* en la práctica. Conviene usar: $\text{abs}(i - k) < \text{eps}$.

Tipos de datos

Valores de verdad (bool):

Hay dos valores de verdad posibles: “verdadero” (*true*) y “falso” (*false*).

Operaciones de booleanos:

Operador C++	Operación
!	Negación
&&	Conjunción
	Disyunción

Tablas de verdad:

p	!p
true	false
false	true

p	q	p && q	p q
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Tipos de datos

Arreglo (*array*):

Un arreglo es una **colección de valores** (o *elementos*).

Se accede a cada valor mediante un **índice** (entero ≥ 0).

Todos los valores son de un **mismo tipo**: p.ej., arreglo de enteros.

45	657	-56	4	23	-5	0	113
0	1	2	3	4	5	6	7

Los índices de un arreglo de N elementos **no** van de 1 a N, sino **de 0 a N-1**.

Tipos de datos

Operaciones de arreglos:

Operador C++	Operación
<code>array <T, n> a;</code>	Crea un arreglo de tipo T y tamaño n .
<code>a[i]</code>	i -ésimo elemento del arreglo a .
<code>a.size()</code>	Longitud del arreglo a .

Para usar el tipo `array` en C++, incluir al principio:

```
#include <array>
using namespace std;
```

Nota: Hay otras formas de trabajar con arreglos y tipos parecidos en C++. En la materia elegimos **`std::array`**, que nos parece la más sencilla de aprender.

Tipos de datos

Cadena de caracteres (**string**):

Un **caracter** (**char**) es un símbolo válido en la computadora:

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

1234567890

!@#\$%*()-_+=~`':;,."<>?/

etc.

En C++ se escriben entre comillas simples: 'a'.

Un **string** es una cadena o secuencia de caracteres.

Nota: Hay varias formas de trabajar con strings en C++.

En la materia elegimos **std::string**, que nos parece la más sencilla de aprender.

Tipos de datos

Operaciones de strings:

Operador C++	Operación
s.size()	Devuelve la longitud del string s.
s[i]	Devuelve el i-ésimo caracter del string s.
< <= == > >=	Compara dos strings. Ej: s1 <= s2
+	Pega dos cadenas. Ej: s1 + s2
...	...

Para usar el tipo string en C++, incluir al principio:
#include <string>
using namespace std;

Tipos de datos - Resumen

Tipo de datos	Ejemplos
bool	true, false
int	3, 0, -5
float, double	3.0, 0.0, -5.0, 3.141592
array	[10, 20, 30], ['a', 'b', 'c']
string	"pepe", "coco"

Expresión

Una **expresión** es una combinación de literales, variables y operadores.

La **evaluación** de una expresión arroja como resultado un valor.

Ejemplos:

¿Qué valores resultan de evaluar estas expresiones (suponiendo que s es un string con valor “hola”)?

1

s.size() + 6

(1>0) || !('a'<'b')

(5.6 > 2.0) && (s.size() < 2)

Un **literal** es un valor particular utilizado directamente en el código del programa. En los ejemplos de arriba: **1 6 1 0 'a' 'b' 5.6 2.0 2**

Asignación

VARIABLE = EXPRESIÓN ;

Almacena el valor de la *EXPRESIÓN* en la dirección en memoria denotada por *VARIABLE*.

Ejemplos:	x = 1000;	// Bien.
	1000 = x;	// Mal. 1000 no es una variable.
	x = y;	// Bien.
	x = x;	// Bien. Aunque no tiene efecto.
	x = x + y * 22;	// Bien.
	x + 1 = y;	// Mal. x + 1 no es una variable.

Ejemplo:

Una expresión también puede incluir llamadas a funciones:

```
#include <iostream>
```

```
#include <cmath> // incluye seno, coseno, tangente, etc.
```

```
using namespace std;
```

```
int main() {
```

```
float x = 2 + 5;
```

```
float y = sin(x) + cos(x);
```

```
cout << y;
```

```
return 0;
```

```
}
```


Entrada y Salida desde Consola

Los programas C pueden recibir/imprimir datos desde/a la consola (teclado y monitor) para interactuar con una aplicación.

- `cout`: console out. Imprime por pantalla un dato (usamos `<<`)
- `cin`: console in. Lee un dato de teclado. (usamos `>>` y hay que apretar ENTER)

Ejemplo

```
#include <iostream>
using namespace std;
int main() {
    cout << "Ingrese un valor entero" << endl;
    int valor = 0;
    cin >> valor;
    cout << "El valor ingresado fue " << valor << endl;
    return 0;
}
```

Secuencialización

Un **programa** es una secuencia finita de **sentencias**.

Si *PROG1* y *PROG2* son programas, entonces

PROG1 PROG2

también es un programa.

Se ejecuta primero *PROG1*. Al terminar, se ejecuta *PROG2*.

Ejemplo:

```
int a;
```

```
a = 10;
```

```
cout << "La variable a tiene valor " << a << endl;
```

Estado

Se denomina **estado** al valor de todas las variables de un programa en un punto de su ejecución.

**Es una “foto” de la memoria
en un momento determinado.**

Estado

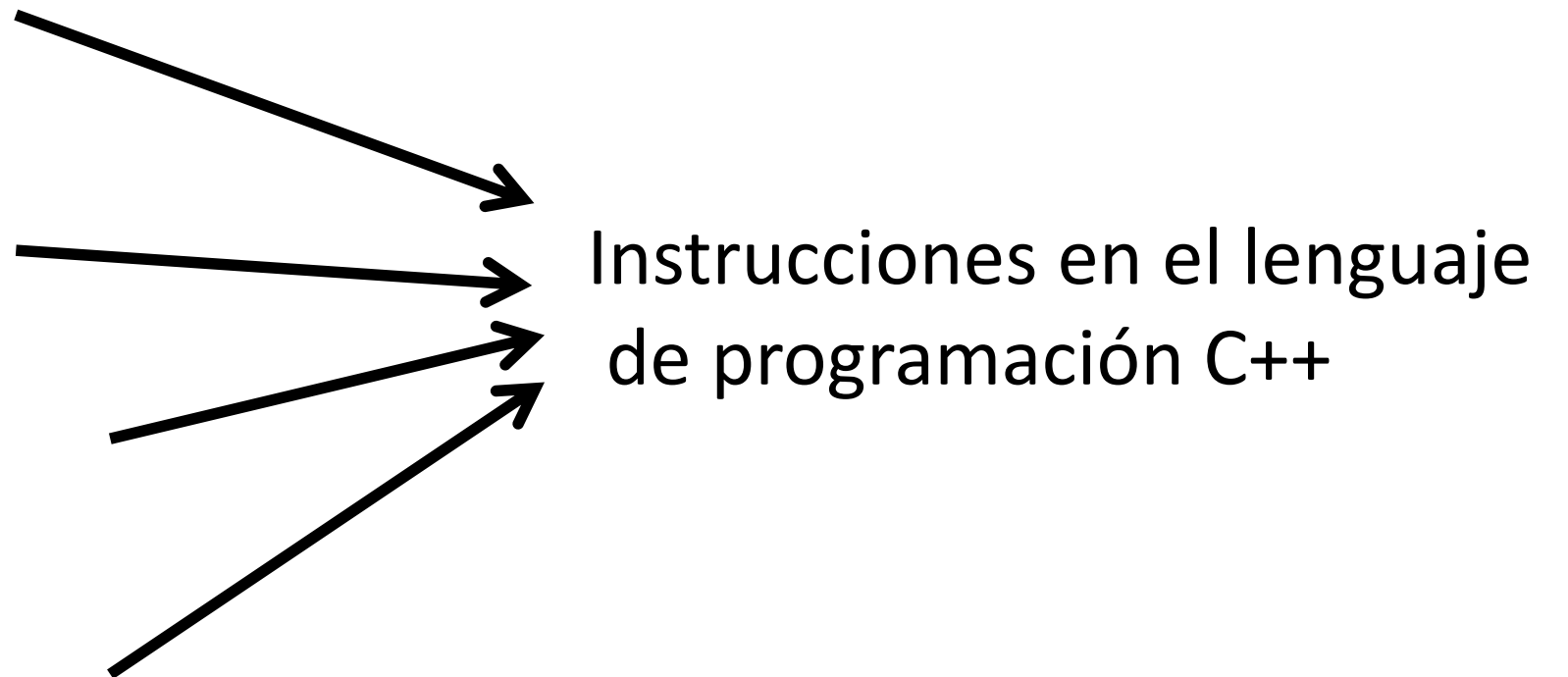
Ejemplo:

```
int x, y;
```

```
y = 10;
```

```
x = y * 2;
```

```
y = y + 1;
```



Estado

Ejemplo:

Descripción de los estados
del programa en lógica de
predicados

int x, y;



$\{ x=\uparrow \wedge y=\uparrow \}$

y = 10;



$\{ x=\uparrow \wedge y=10 \}$

x = y * 2;



$\{ x=20 \wedge y=10 \}$

y = y + 1;



$\{ x=20 \wedge y=11 \}$

\uparrow significa “valor indefinido”

Otro Ejemplo:

int a;

array<int, 2> b;

$\{ a=\uparrow \wedge |b|=2 \wedge b[0]=\uparrow \wedge b[1]=\uparrow \}$

a = 0;

$\{ a=0 \wedge |b|=2 \wedge b[0]=\uparrow \wedge b[1]=\uparrow \}$

b[a] = 1;

$\{ a=0 \wedge |b|=2 \wedge b[0]=1 \wedge b[1]=\uparrow \}$

b[b[0]] = 999 * b[a] + a;

$\{ a=0 \wedge |b|=2 \wedge b[0]=1 \wedge b[1]=999 \}$

b[b[1]/333-2] = 123;

$\{ a=0 \wedge |b|=2 \wedge b[0]=1 \wedge b[1]=123 \}$

Condicional

if (CONDICIÓN) { PROG1 }

CONDICIÓN es una expresión que arroja resultado verdadero o falso;
PROG1 es un programa.

PROG1 se ejecuta **si y sólo si** *CONDICIÓN* arroja valor verdadero.

Ejemplo:

```
if (1 > 5) {  
    cout << "1 es mayor que 5." << endl;  
}  
if (1 < 5) {  
    cout << "1 es menor que 5." << endl;  
}
```


Condicional

if (*CONDICIÓN*) { *PROG1* } else { *PROG2* }

CONDICIÓN es una expresión que arroja V o F.
PROG1 y *PROG2* son programas.

PROG1 se ejecuta **sii** *CONDICIÓN* arroja valor verdadero.
PROG2 se ejecuta **sii** *CONDICIÓN* arroja valor falso.

Ejemplo:

```
if (1 > 5) {  
    cout << "1 es mayor que 5." << endl;  
} else {  
    cout << "1 es menor que 5." << endl;  
}
```

Condicional – Otro ejemplo

¿Qué imprime por pantalla este código?

```
int a = 10;  
array<int, 2> b = {100, 1};  
  
if (b[0] / (a * 10) == b[1]) {  
    b[0] = b[0] - 1;  
    b[1] = b[1] * 5;  
} else {  
    b[0] = b[0] + 1;  
    b[1] = b[1] * 3;  
}  
cout << a << "," << b[0] << "," << b[1];
```



Para mayor claridad,
indentar cada bloque de código.

Ciclo

while (*CONDICIÓN*) { *PROG1* }

CONDICIÓN es una expresión que arroja resultado verdadero o falso;
PROG1 es un programa.

PROG1 se ejecuta una y otra vez, **mientras** *CONDICIÓN* siga arrojando valor verdadero.

Ejemplo:

```
int i = 0;
while (i < 3) {
    cout << i;
    i = i + 1;
}
```

Ciclo – Otro ejemplo

while (CONDICIÓN) { PROG1 }

Ejemplo:

```
int i = 0;
while (i < 3) {
    if (i % 2 == 0) {
        cout << i << " es par" << endl;
    } else {
        cout << i << " es impar" << endl;
    }
    i = i + 1;
}
```

Ciclos anidados

¿Qué imprime por pantalla este código?

Ejemplo:

```
int fil = 1;
while (fil <= 5) {
    int col = 1;
    while (col <= fil) {
        cout << col;
        col = col + 1;
    }
    cout << endl;
    fil = fil + 1;
}
```





Alcance de las variables

El **alcance** (*scope*) de una variable es el código en el cual una variable puede ser accedida.

Su definición varía según el lenguaje.

En C++ el alcance de una variable va:

- desde su **definición**,
- hasta el final del **bloque actual**,
- incluyendo **bloques anidados**.

```
main() {  
    a = 1;   
    int i = 0;  
    while (i < 10) {  
        int a = 1;  
        if (i > 5) {  
            i = i + a;   
        }  
        cout << i + a;   
        i = i + 1;  
    }  
    cout << a;   
}
```

Evaluación de expresiones lógicas

```
array<int, 3> a = {2, 4, 8};  
int i = 0;  
while (i<a.size() && a[i]%2==0) {  
    cout << a[i];  
    i = i + 1;  
}
```

Evaluación *Lazy*:

false && *EXP* → false

true || *EXP* → true

En estos casos, la expresión lógica *EXP* **no se evalúa**.

Otros ejemplos:

¿Cuánto vale (a!=0 && 1/a>0.1) si a=0? ¿Si a=1?

¿Cuánto vale (a==0 || 1/a>0.1) si a=0? ¿Si a=1?

Azúcar sintáctica en C++

Sintaxis añadida para facilitarle las cosas al programador.

NO recomendamos usarla en esta materia.

Azúcar sintáctica en C++ (no recomendada)	Sintaxis recomendada
a++;	a = a + 1;
a--;	a = a – 1;
if (<i>CONDICIÓN</i>) <i>INSTRUCCIÓN</i>;	if (<i>CONDICIÓN</i>) { <i>INSTRUCCIÓN</i>; }
for (int i=0; i<10; i++) { <i>PROGRAMA</i> }	int i = 0; while (i < 10) { <i>PROGRAMA</i>; i = i + 1; }

Programa (resumen)

Un **programa** es una secuencia de **sentencias**.

- Declaración de variables

TIPO NOMBRE;

- Asignación

VARIABLE = EXPRESIÓN;

- Condicional

if (CONDICIÓN) { PROG1 } else { PROG2 }

- Ciclo

while (CONDICIÓN) { PROG1 }

Repaso de la clase de hoy

- Condicional: `if.. ; if..else..`
- Ciclo: `while..`; ciclos anidados.
- Evaluación lazy de expresiones booleanas.

Próximos temas

- Modularidad del código: funciones.
- Especificación de problemas.
- Correctitud de programas.