

Introducción a la Computación (para Matemáticas)

Primer Cuatrimestre de 2018



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Introducción a la Computación (para Matemáticas)

Primer Cuatrimestre de 2018

Docentes: Ricardo Oscar Rodriguez (Profesor)
Gervasio Pérez (JTP)
Mariano Rean (Ayudante de 1ra)
Ignacio Mollo (Ayudante de 2da)

Clase de hoy:

- Cuestiones administrativas: horarios, mails, Campus, etc.
- Objetivos, correlatividades, evaluación, bibliografía.
- Repaso superficial del contenido de toda la materia.
- Clase práctica: setup de cuentas, Ubuntu, CLion.

Introducción a la Computación (para Matemáticas)

Primer Cuatrimestre de 2018

Horarios y lugar: Lunes y jueves de ¿14 a 18 horas?

Laboratorio 4 (en el subsuelo del Depto. de Computación).

En general: 1) teórica, 2) práctica.

Listas de correo:

icm-doc@dc.uba.ar → para escribir sólo a docentes.

icm-alu@dc.uba.ar → para comunicarse con docentes y alumnos.

Página Campus: <https://campus.exactas.uba.ar/course/view.php?id=1010>

Participación en clase

Esperamos una activa participación en clase:

- No hay preguntas tontas.
- Si una explicación no responde a tu pregunta, por favor, volver a preguntar.
- Compartí tus dudas. Tus certezas también.
- Las interrupciones pertinentes no molestan, sobre todo si son para marcar errores, hacer comentarios o reencauzar la clase.
- No está permitido permanecer callado en la resolución conjunta de ejercicios.

Filosofía de enseñanza

- Nosotros tenemos la *obligación* de enseñar.
- Ustedes tienen el *derecho* de aprender.
- No somos los dueños del conocimiento y no lo sabemos todo, así que, quizás más de una vez la respuesta sea “*No lo se*” y otras podemos equivocarnos.
- Defendemos la Universidad Pública, Masiva, Laica y Gratuita. A veces, por eso, salimos a la calle y suspendemos clases.

Filosofía de enseñanza

Enseñar no es transmitir
conocimientos, sino
crear las posibilidades
para su producción
o su construcción.

Paulo Freire

Introducción a la Computación (para Matemáticas)

Objetivos:

- Al finalizar el curso, se espera que el alumno pueda especificar, diseñar y programar algoritmos, que resuelvan problemas de tamaño pequeño a mediano.
- Se pretenderá que los alumnos dominen elementos básicos de: programación, especificación de problemas, correctitud de programas, técnicas algorítmicas, cómputo de complejidad, y tipos abstractos de datos. En particular se mostrarán algunos ejemplos paradigmáticos de resolución de problemas de ordenamiento y búsqueda.

O al menos sepan que...



"Cuando algo sale mal, basta con apretar este pequeño botón y reiniciar. Quisiera que todo en la vida fuera así"

Introducción a la Computación (para Matemáticas)

¿Quiénes pueden cursar la materia?

- La materia es obligatoria para la orientación aplicada de la Licenciatura en Cs. Matemáticas.
- Correlatividades (alumnos de Matemáticas):
 - TPs de “Análisis II” y “Elementos de cálculo numérico”.
 - Final de “Análisis I” y “Algebra I”.
- Alumnos de otras carreras pueden cursar la materia, aunque necesitarán manejar algunos conocimientos necesarios.
- **Introducción a la Computación (para Biología y otras carreras)**
 - Profesor: Esteban Mocskos
 - Se dicta los segundos cuatrimestres.

Introducción a la Computación (para Matemáticas)

Algunos conocimientos necesarios:

- Números naturales y enteros.
Números primos, coprimos, Fibonacci, etc.
- Lógica de primer orden.
 $(\forall x) (\text{Primo}(x) \Rightarrow (\neg \exists y) (x \bmod y = 0))$
- Principio de inducción.
i) $P(0)$; ii) $P(n) \Rightarrow P(n+1)$
- Recursión, ecuaciones de recurrencia.
 $a_0 = 1; a_n = 2 a_{n-1} \quad (n \geq 2) \quad \rightarrow \quad a_n = 2^n$

Introducción a la Computación (para Matemáticas)

Modo de evaluación:

- Dos exámenes parciales individuales.
- Tres trabajos prácticos grupales.
- La materia se promociona si $(P1+P2)/2 \geq 7$ y los tres TPs están aprobados.

Bibliografía:

- Balcazar, "Programación metódica", McGraw-Hill, 1993.
- Dijkstra, "A discipline of programming", Prentice Hall, 1973.
- Aho, Hopcroft & Ullman, "Estructuras de Datos y Algoritmos", A-W, 1988.
- Cormen, "Introduction to Algorithms", MIT Press, 2009.
- Kernighan & Ritchie, El lenguaje de programación C, Prentice Hall, 1991.
- Stroustrup, "The C++ Programming Language", Addison-Wesley, 1997.
- Elkner, Downey & Meyers, "How to Think Like a Computer Scientist",
<http://www.openbookproject.net/thinkcs/python/english2e/>
- Tutorial online de Python, <http://docs.python.org/tutorial/>

Introducción a la Computación (para Matemáticas)

¿Qué es programar?

- Programar \neq Manejar un lenguaje de programación.
- Especificación formal, correctitud, eficiencia, modularidad, usabilidad, adaptabilidad, ...
- **Materia muy amplia.**
 - Teóricas cortas, seguidas de práctica/taller.
 - Fuerte carga de ejercitación (en el labo y en casa).

Introducción a la Computación (para Matemáticas)

- 1) Elementos básicos de programación**
- 2) Especificación y correctitud**
- 3) Algoritmos de búsqueda y ordenamiento**
- 4) Tipos abstractos de datos**
- 5) Técnicas algorítmicas**

Introducción a la Computación (para Matemáticas)

1) Elementos básicos de programación

- Tipos de datos: enteros, reales, strings, etc.
- Variables y expresiones.
- Instrucción: asignación, condicional, ciclo.
- Estado de un programa.
- Funciones, pasaje de parámetros.

2) Especificación y correctitud

3) Algoritmos de búsqueda y ordenamiento

4) Tipos abstractos de datos

5) Técnicas algorítmicas

Introducción a la Computación (para Matemáticas)

1) Elementos básicos de programación

2) Especificación y correctitud

- ¿Qué debe hacer un programa?
- ¿Un programa hace lo que se supone que debe hacer?

3) Algoritmos de búsqueda y ordenamiento

4) Tipos abstractos de datos

5) Técnicas algorítmicas

Introducción a la Computación (para Matemáticas)

1) Elementos básicos de programación

2) Especificación y correctitud

Fin de la primera mitad. Primer parcial: 3-5-18

3) Algoritmos de búsqueda y ordenamiento

4) Tipos abstractos de datos

5) Técnicas algorítmicas

Introducción a la Computación (para Matemáticas)

1) Elementos básicos de programación

2) Especificación y correctitud

3) Algoritmos de búsqueda y ordenamiento

- Buscar un elemento en un arreglo.
- Ordenar los elementos de un arreglo.
- Conceptos de recursión y complejidad.

4) Tipos abstractos de datos

5) Técnicas algorítmicas

Introducción a la Computación (para Matemáticas)

- 1) Elementos básicos de programación**
- 2) Especificación y correctitud**
- 3) Algoritmos de búsqueda y ordenamiento**
- 4) Tipos abstractos de datos**
 - Lista, cola, pila, árbol, diccionario, etc.
- 5) Técnicas algorítmicas**

Introducción a la Computación (para Matemáticas)

- 1) Elementos básicos de programación**
- 2) Especificación y correctitud**
- 3) Algoritmos de búsqueda y ordenamiento**
- 4) Tipos abstractos de datos**
- 5) Técnicas algorítmicas**
 - Divide & conquer
 - Backtracking
 - (Heurísticas)

Introducción a la Computación (para Matemáticas)

- 1) Elementos básicos de programación**
- 2) Especificación y correctitud**
- 3) Algoritmos de búsqueda y ordenamiento**
- 4) Tipos abstractos de datos**
- 5) Técnicas algorítmicas**

Fin de la segunda mitad. Segundo parcial : 5-7-18

Recuperatorio 1ro. : 12-7-18

Recuperatorio 2do.: 19-7-18

Resolución de problemas Informal

- Requiere una **descripción/especificación** del dominio de aplicación y de las características de las soluciones buscadas. Por ej. Suma de las cardinalidades de dos conjuntos finitos.
- Luego es necesario encontrar/diseñar un **algoritmo** que lo resuelva. Por ej. Suma escolar.
- Finalmente debemos **programar** el algoritmo.

Sumar dos cardinalidades

- **Descripción:** Dadas las cardinalidades de dos conjuntos finitos (representadas por números naturales) se busca encontrar una tercera (otro número natural) que represente la cardinalidad del conjunto unión.

Abstracción: Concepto de número y operación básica de suma

Algoritmo de Sumar:

Para los número entre 0 a 9 podemos hacer:

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

¿Podemos tabular todo?

Siguiendo pasos:

- Poner los dos números uno debajo del otro...
- Pero así no

645	sino así	645
1241		1241
- Ahora colocar debajo de cada columna en una tercera fila el número indicado en la tabla.
- El número obtenido es el resultado buscado.

¿Funciona siempre?

Siguiendo pasos 2

- Debemos incluir la regla del acarreo. Si el número que debemos poner según la table es mayor que 9, entonces el resultado de la siguiente columna a la izquierda debe ser incrementado en 1.

Algunas cuestiones:

- ¿Este algoritmo resuelve el problema? ¿Cómo lo determino?
- ¿Este algoritmo es único? ¿Hay alternativas?
- ¿Cuáles son las operaciones primitivas?

Algoritmo

- Sucesión de pasos que al ser ejecutados metódicamente en el orden prescripto una cantidad finita de veces nos da el resultado buscado.
- Por ejemplo el algoritmo para encontrar el promedio entre dos números naturales podría ser:
 1. Sumar los dos números.
 2. Dividir el resultado por dos.
- Otro ejemplo de algoritmos son: las recetas de cocina, instrucciones de uso o de ensamblaje, construcciones geométricas (bisección de un triángulo).

Algunas cuestiones

Al analizar un algoritmo tendremos en cuenta algunas cuestiones tales como:

- Su correspondencia con la descripción original
¿Siempre da respuestas correctas? ¿Resuelve todas las instancias?
- Su eficiencia en el uso de recursos como tiempo y espacio.

Resolución de problemas Informal

- Determinar si un número natural es primo.
- Especificación?
- Algoritmo?
- Programa?

Resolución de problemas Informal

- Determinar si un número natural es primo.
- Especificación:

$$\textit{Primo}(x) \equiv \nexists \ 1 < y < x: x \bmod y = 0$$

- Algoritmo?
- Programa?

Resolución de problemas Informal

- Determinar si un número natural es primo.
- Especificación:

$$\textit{Primo}(x) \equiv \nexists \ 1 < y < x: x \bmod y = 0$$

- Algoritmo? Idea: probar con todos los números.
- Programa?

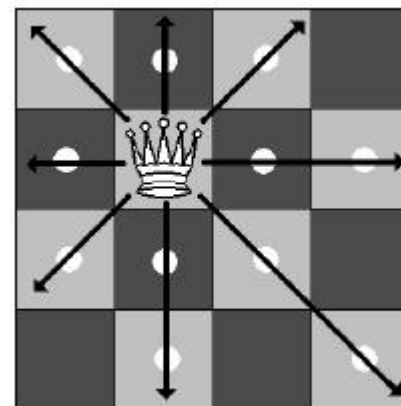
Introducción a la Computación (para Matemáticas)

Dos problemas para pensar:

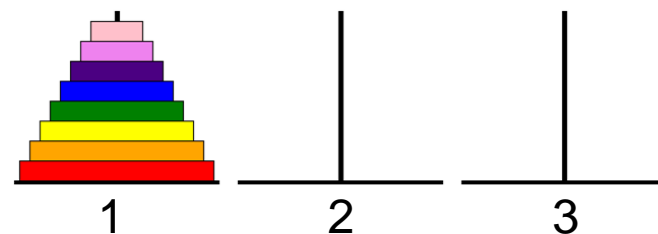
- Colocar 8 reinas en un tablero de ajedrez sin que se amenacen.

(Tablero: 8x8.)

<http://spaz.ca/aaron/SCS/queens/>



- Torre de Hanoi:
Mover N discos de la estaca 1 a la 3.
Mover de a un disco por vez.
No se puede colocar un disco sobre otro de menor tamaño.



<http://www.cut-the-knot.org/recurrence/hanoi.shtml>

Introducción a la Computación (para Matemáticas)

Pregunta frecuente:

- ¿Qué lenguaje de programación vamos a usar?

Respuesta corta:

- En la primera mitad, C++. En la segunda mitad, Python.

Respuesta larga:

- No importa demasiado. Lo que más importa son los conocimientos básicos de programación, que son comunes a la mayoría de los lenguajes.

Introducción a la Computación (para Matemáticas)

Analogía con los Lenguajes Naturales:

- Los padres enseñan a sus hijos a interactuar socialmente:
 - saludar al llegar y al irse;
 - pedir cosas;
 - agradecer;
 - preguntar y responder;
 - ...
- Estas acciones son **independientes** del lenguaje usado (español, alemán, japonés, suajili, etc.).
- Al aprender un lenguaje nuevo, no necesitamos que nos expliquen esos conceptos.



Introducción a la Computación (para Matemáticas)

Objetivo de esta materia:

- Que incorporen los elementos básicos de programación, que son **independientes** del lenguaje usado (Python, C++, Perl, Visual Basic, Pascal, Java, Fortran, Cobol, etc.).
- Que en el futuro, al aprender un lenguaje nuevo, no necesiten que les expliquen esos conceptos.



Algunas nociones más

- Una computadora es una máquina de propósito general.
- Programar es construir un puente entre la computadora y una aplicación específica.
- En términos abstractos una computadora es una máquina de estados.
- Entonces programar es escribir un texto formal que transforme un estado inicial en uno final.
- Dicho texto es una secuencia de sentencias donde cada uno de sus elementos juega un rol preciso.

Noción de estado. 1

Pensemos en una cafetera eléctrica:



¿Cuál es el estado inicial?

¿Cuál es el estado final?

Noción de estado. 2

- Los estados serán conjuntos de variables que describen un problema.
- Un estado es una instancia particular de dichas variables.
- Programar será identificar todas las variables involucradas en la resolución de un problema y dar una secuencia finita de instrucciones que permita transformar estados iniciales en finales.

Noción de estado. 3

- El problema de Sumar involucra 3 variables.
- El problema de Primo 2 variables.

Resumen

- La resolución de problemas involucra:
 - Especificar.
 - Encontrar un algoritmo.
 - Programar.
- A su vez programar requiere identificar las variables implícitas del algoritmo y dar la secuencia finita de operaciones básica que lo implementa.

Programa

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

Ejemplo:

- 1.- Moje el cabello.
- 2.- Coloque shampoo.
- 3.- Masajee suavemente y deje actuar por 2 min.
- 4.- Enjuague.
- 5.- Repita el procedimiento (desde 1.-).

Programa

- **Otro 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 el *estado*, o bien
- modifica el flujo de ejecución.

Instrucción

Una **instrucción** es una operación que:

- transforma el *estado*, o bien
- modifica el flujo de ejecución.

- 1.- Moje el cabello.
- 2.- Coloque shampoo.
- 3.- Masajee suavemente y deje actuar por 2 min.
- 4.- Enjuague.
- 5.- Repita el procedimiento (desde 1.-)

Instrucción

Una **instrucción** es una operación que:

- transforma 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

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

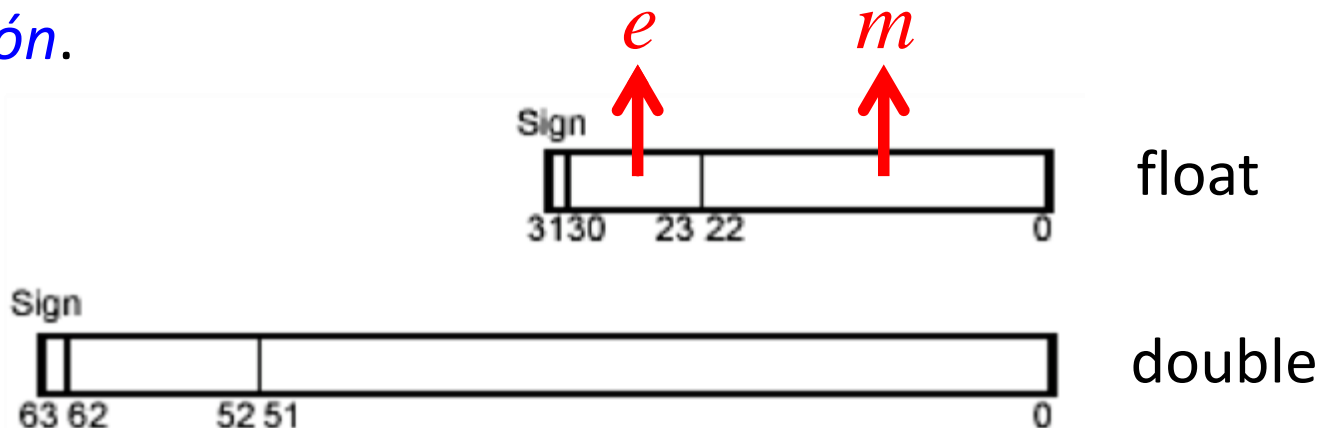
- **Reales** (**float** y **double** en C++):

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

- $f \approx \pm m * 10^e$ 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"

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)
- ...

Variable

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

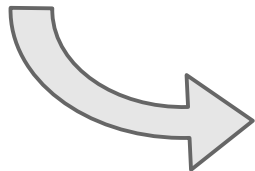
En esa celda de memoria es posible:

- **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;
```

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*.

x = 1000; // **Bien.**

1000 = x; // **Mal.** 1000 no es una variable.

- Ejemplos: **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.

Secuencialización

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

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

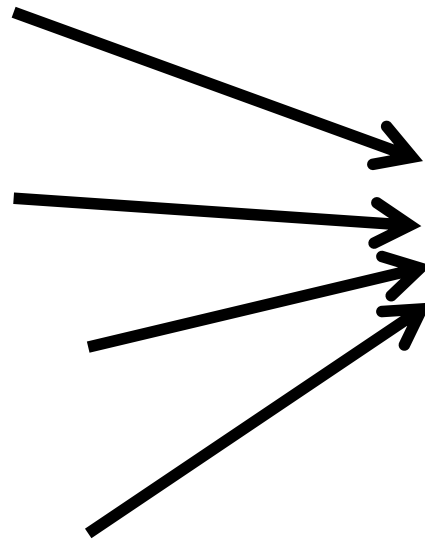
Ejemplo:

```
int x, y;
```

```
y = 10;
```

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

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



Instrucciones en el lenguaje de

Estado

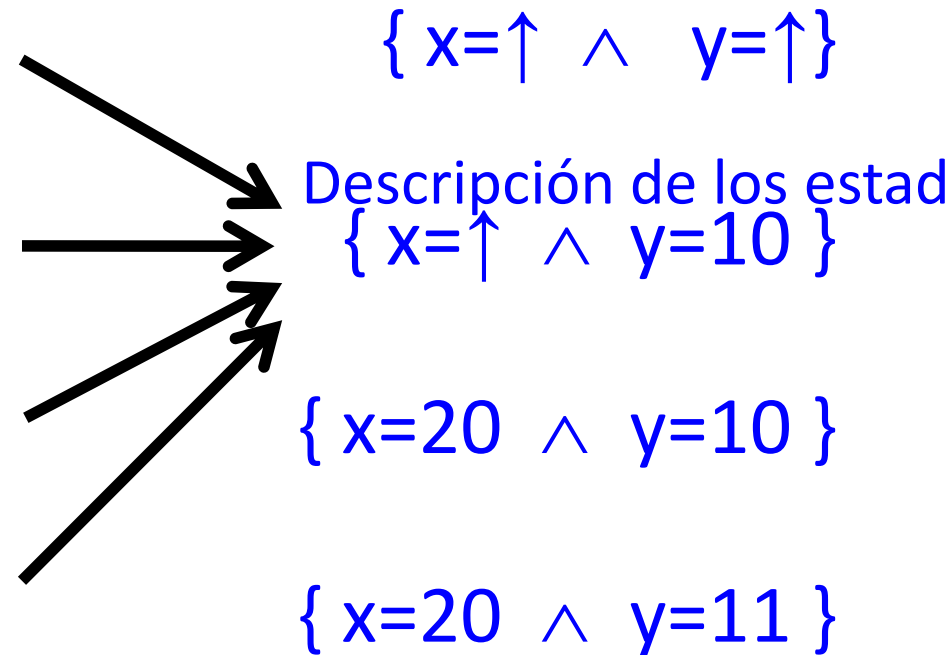
Ejemplo:

```
int x, y;
```

```
y = 10;
```

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

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



\uparrow significa "valor indefinido"

Repaso de la clase de hoy

- Valor. Tipos de datos: bool, int, float, string, array.
- Expresiones, variables, literales.
- Memoria, estado.
- Programa, instrucción, asignación, secuencialización.

.Próximos temas

- Condicionales, ciclos, funciones.