Tipos de datos

- **int** (Números enteros)
- **float** (Números reales)
- **double** (Números reales más grandes que float)
- **bool** (Valores lógicos)
- **char** (Caracteres y cualquier cantidad de 8 bits)
- **void** (Nada. Sirve para indicar que una función no devuelve valores)

int × int → int	
+	Suma
-	Resta
*	Producto
/	División entera
%	Resto de la división entera (módulo)
- , +	Signo negativo, positivo
++	Incrementación
--	Decrementación

int → int

double × double → double	
+	Suma
-	Resta
*	Producto
/	División en coma flotante
- , +	Signo negativo, positivo
++	Incrementación
--	Decrementación

double → double

Algunas funciones reales

abs: int → int Calcula el valor absoluto de un número

ceil: double → double Calcula el número entero mayor o igual que el dado

floor: double → double Redondea por defecto el valor de un número

sqrt: double → double Calcula la raíz cuadrada de un número

pow:double × double → double Calcula la potencia de un número

Algunas funciones de tipo carácter :cctype

isdigit: char → bool Devuelve TRUE si el carácter es: '0', … ,'9'

isalpha: char → bool Devuelve TRUE si el carácter es: 'A', … ,'Z', 'a', … , 'z'.

islower: char → bool Devuelve TRUE si el carácter es una letra minúscula: 'a', … , 'z'.

isupper: char → bool Devuelve TRUE si el carácter es una letra mayúscula: 'A', … ,'Z'.

tolower: char → char Convierte un carácter mayúscula en minúscula.

toupper: char → char Convierte un carácter minúscula en mayúscula.

Operadores (prioridad)

++ -- (postfijos)
Llamadas a funciones
Moldes

++ -- (prefijos) !
- (cambio de signo)

*** / %**

+ -

< <= > >=

== !=

&&

||

?:

= += -= *= /= %=

Variables y constantes

Una variable es un espacio reservado en la memoria del ordenador para contener valores que pueden cambiar durante la ejecución de un programa.

```
<tipo_de_dato>  <nombre_de_variable> ;

<tipo_de_dato>  <lista de variables separadas por comas> ;

<tipo_de_dato>  <nombre_de_variable> = valor;
```

Ejemplos

```
int  x;
float num1, num2, suma=0;
char  letra='A';
bool esPrimo=true, enc=false;
```

Una constante es un objeto cuyo valor no puede ser modificado

```
const  <tipo_de_dato>   <nombre_de_constante> =  <valor> ;
```

Ejemplos

```
const bool OK = true;
const char SONIDO = '\a';
const int MAXIMO = 5000;
const double PI = 3.141592;
```

Identificadores (nombres)

- No pueden contener espacios en blanco, ni ciertos caracteres especiales como letras acentuadas, la letra eñe, las barras \ o /, etc. El compilador determina la máxima longitud que pueden tener (por ejemplo, 31 caracteres)
- Sensibles a mayúsculas y minúsculas.
- No se podrá dar a un dato el nombre de una palabra reservada. No es recomendable usar el nombre de algún identificador usado en las bibliotecas estándar (por ejemplo, cout)
- El identificador de un dato debe reflejar su semántica (contenido).
- Usaremos minúsculas, salvo la primera letra de nombres compuestos.
- No se nombrarán dos datos con identificadores que difieran únicamente en la capitalización, o un sólo carácter. (Num y num)

Funciones interesantes en math.h ó cmath

`double sin(double r);`	seno, $\sin r$ (en radianes)
`double cos(double r);`	coseno, $\cos r$ (en radianes)
`double tan(double r);`	tangente, $\tan r$ (en radianes)
`double asin(double x);`	arco seno, $\arcsin x, x \in [-1, 1]$
`double acos(double x);`	arco coseno, $\arccos x, x \in [-1, 1]$
`double atan(double x);`	arco tangente, $\arctan x$
`double atan2(double y, double x);`	arco tangente, $\arctan y/x$
`double sinh(double r);`	seno hiperbólico, $\sinh r$
`double cosh(double r);`	coseno hiperbólico, $\cosh r$
`double tanh(double r);`	tangente hiperbólica, $\tanh r$
`double sqrt(double x);`	$\sqrt{x}, x \geq 0$
`double pow(double x, double y);`	x^y
`double exp(double x);`	e^x
`double log(double x);`	logaritmo neperiano, $\ln x, x > 0$
`double log10(double x);`	logaritmo decimal, $\log x, x > 0$
`double ceil(double x);`	menor entero $\geq x$, $\lceil x \rceil$
`double floor(double x);`	mayor entero $\leq x$, $\lfloor x \rfloor$
`double fabs(double x);`	valor absoluto de x, $\|x\|$
`double ldexp(double x, int n);`	$x2^n$
`double frexp(double x, int* exp);`	inversa de ldexp
`double modf(double x, double* ip);`	parte entera y fraccionaria
`double fmod(double x, double y);`	resto de x/y