VARIABLES Y TIPOS DE DATOS

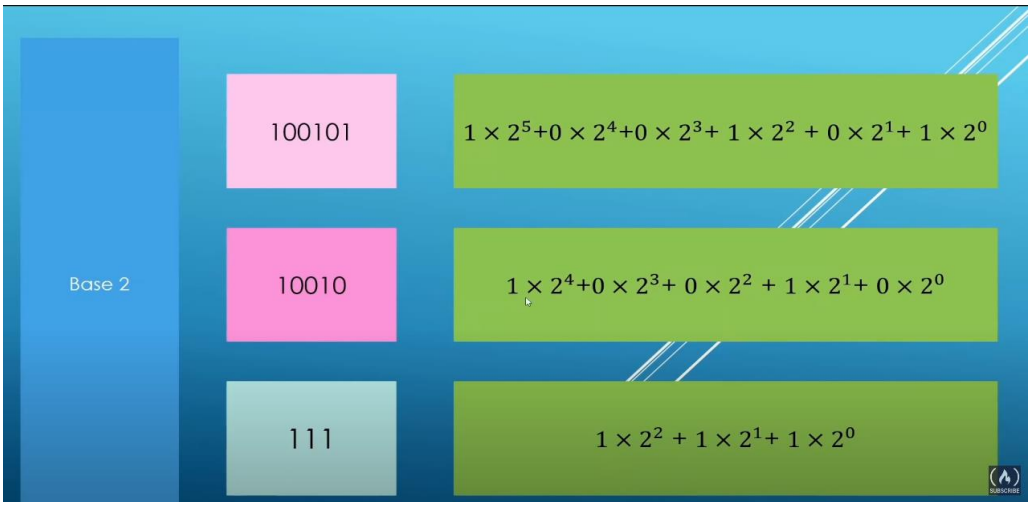
CADA APARTADO PRINCIPAL CORRESPONDE CON EL NOMBRE DEL PROYECTO CON EJEMPLOS DE LO APUNTADO AQUÍ

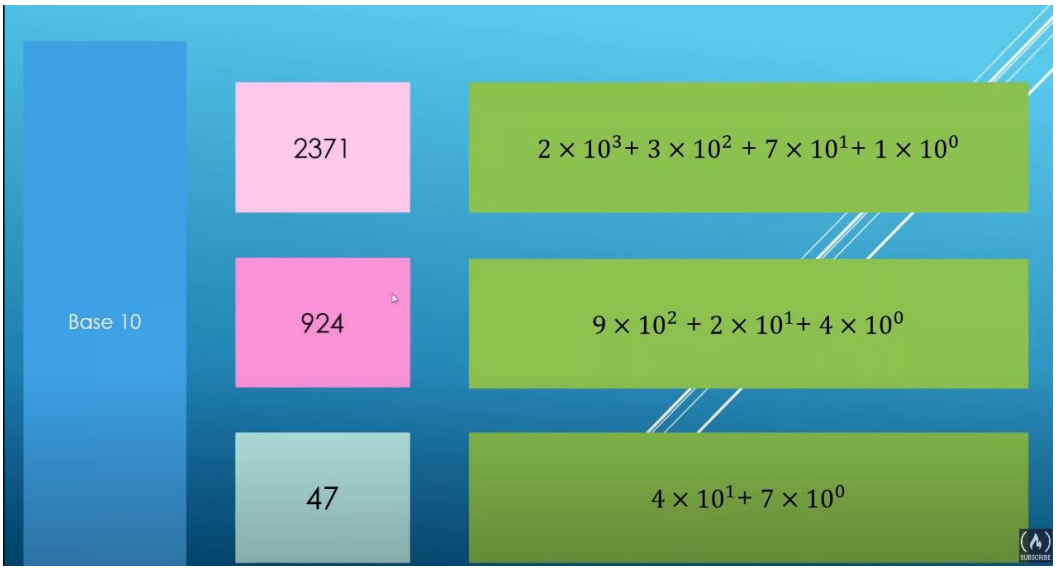
1. INTRODUCCIÓN A LAS VARIABLES Y LOS TIPOS DE DATOS

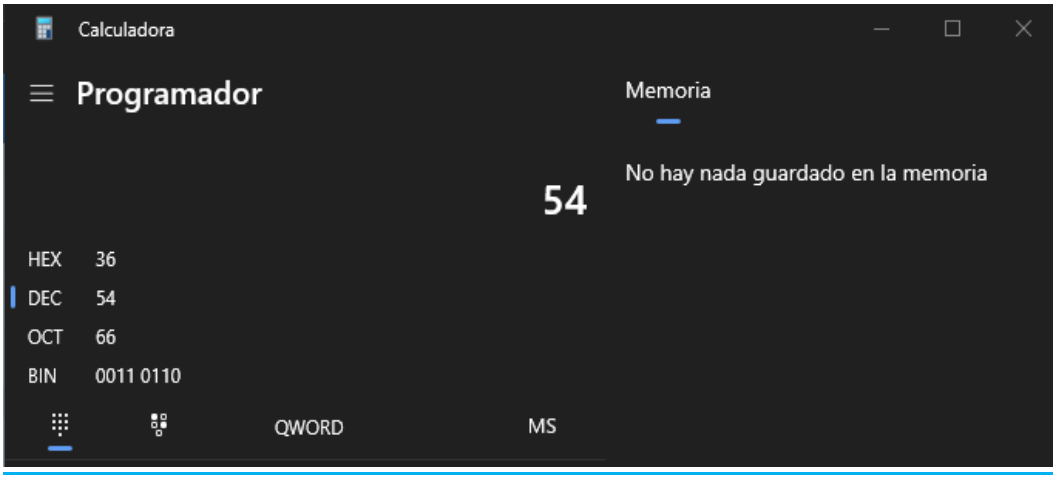
- C++ utiliza varios tipos de datos para almacenar datos
- Los datos almacenados en variables son bits (grupos de celdas de 0 y 1) que se reservan en la RAM para que puedan ser legibles por el ordenador. Estos bits se pueden agrupar en diferentes tamaños, por ejemplo, en 8 (byte) o en 16, etc. Según el rango de datos aumente, así lo hará el número necesario de dígitos para ser representados en memoria.
- Para hacer transformaciones entre las distintas representaciones de datos (que un humano pueda entender los 0 y 1 de la máquina o viceversa) usamos los sistemas numéricos.

2. SISTEMAS NUMÉRICOS

- La diferencia entre los diferentes sistemas numéricos es básicamente la base que se usa al representar el número, es decir, en binario se usa el 2 como base, en decimal (el que los humanos empleamos) se usa el 10, en hexadecimal se usa el 16, etc. Pero son simplemente formas de representar un mismo número.







- El octal y el hexadecimal se usan principalmente para acortar el número de dígitos empleados en la representación de un número en sistema binario.
- Se añade un 0b delante de número binario para indicar que está representado en sistema binario.
- Binario → Octal:
  0110 1110 0011 0000 1111 0001 0011 1111 → 0 15 614 170 477
  Se añade un 0 delante para indicar que el número está representado en el sistema octal.
- Binario → Hexadecimal:
  0110 1110 0011 0000 1111 0001 0011 1111 → 0x 6E30 F13F
  Se añade un 0x delante para indicar que el número está representado en sistema hexadecimal.

3. VARIABLES E INTEGERS

- Se representan en C++ con `int`.
- Se usan para almacenar números enteros.
- Suelen ocupar 4 bytes o más en memoria, pero por lo general suelen ser 4 bytes (32 bits).
- Variable: Una parte de memoria con nombre, empleada para almacenar distintos tipos de datos.
  `typename` nombre Variable `=` valor
- Hay 3 formas de inicializar variables:
  - Inicialización con {}
  - Inicialización con ()
  - Inicialización con =

4. MODIFICADORES DE INTEGERS

- `signed`: Puedes almacenar tanto números positivos como negativos.
- `unsigned`: Solo puedes almacenar números positivos.

Type with modifier	Bytes in memory	Range
unsigned int	4	[0, 4,294,967,295]
signed int	4	[-2,147,483,648, 2,147,483,647]

- `short`: Acorta el rango de valores que puedes almacenar en la variable. Ocupa 2 bytes en memoria.
- `long`: Amplía el rango de valores que puedes almacenar en la variable. Ocupa 8 bytes en memoria.
- Estos modificadores solo funcionan con `int`.

5. NÚMEROS DECIMALES

- Técnicamente se conocen como tipos de coma flotante. Se emplean para representar números con partes fraccionales. Hay 3 tipos y sus diferencias son el tamaño que ocupan en memoria y la precisión, es decir el número de dígitos que puedes representar con este tipo de dato desde el número delante de la coma hasta el último decimal.:
  - `float`: Ocupa 4 bytes. Tiene una precisión de 7. En C++ se usa una ***f*** al final del número para indicar que es un `float`.
  - `double`: Ocupa 8 bytes. Tiene una precisión de 15.

    - `long double`: Ocupa 16 bytes. Tiene una precisión de 15 o más. En C++ se usa una ***L*** al final del número para indicar que es un `long double`
    - Por ejemplo, el número: 1.23456789001 al tener una precisión de 12, requerirá de un `double` para ser representado correctamente.
- Es importante especificar la ***f*** y la ***L*** respectivamente de lo contrario el número será un `double` y no un `float` o un `long`.
- Para representar los números decimales en memoria se usa un sistema numérico especial llamado IEEE_754.
- Por lo general siempre se usará el `double`, ya que el `float` no suele ser suficiente y el `long double` es demasiado.
- Podemos usar números decimales para emplear la notación científica.
- Los tipos de coma flotante te permiten ciertas cosas que los `integers` no:
    - Dividir entre 0, obteniendo así *infinito positivo* o *infinito negativo*, dependiendo si el número por el que divides entre 0 es positivo o negativo.
    - Dividir 0.0/0.0, obteniendo así lo que se llama *NaN* (Not A Number), el programa no crasheará, pero puede haber errores.

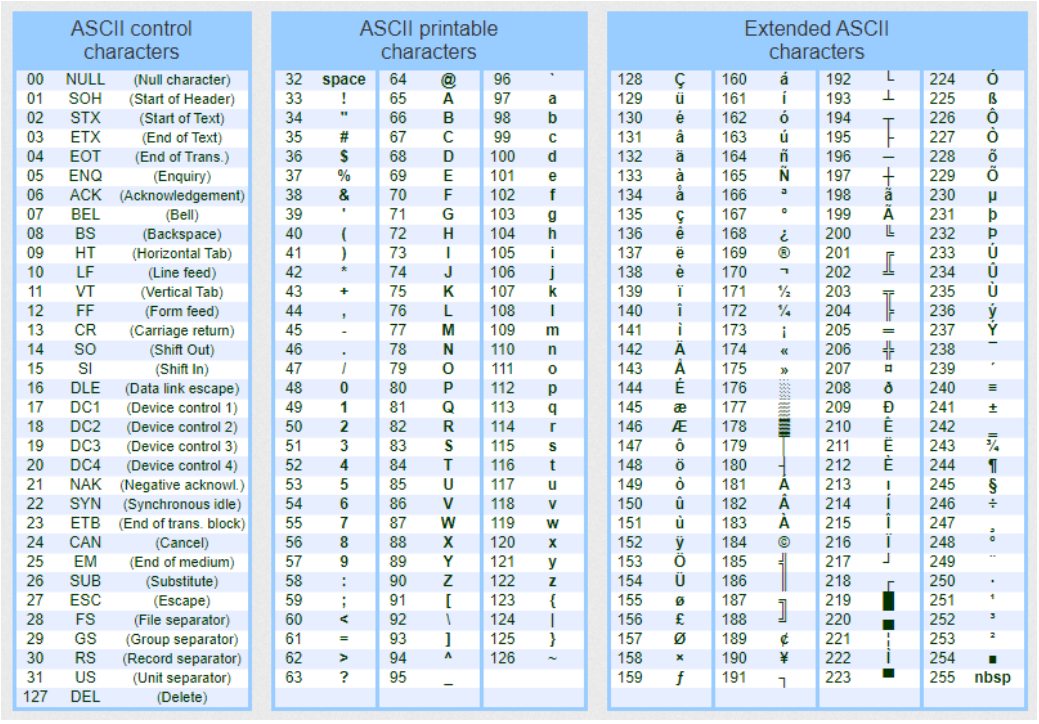
6. BOOLEANS

- Este tipo de dato puede almacenar 2 valores y nos permite tomar decisiones en nuestro programa:
    - `true`
    - `false`
- Ocupan 1 byte en memoria.
- Con `std::cout << std::boolalpha` vemos el valor real de `bool` en vez de 1 y 0 como valor por defecto.

7. CARACTERES Y TEXTO

- Para representar caracteres se usa el tipo de dato llamado `char`.
- Se usa comilla simple para indicar que es un `char`.
- Ocupa 1 byte en memoria.

- Gracias al código ASCII, puedes asociar un único valor de `char` a 256 valores distintos.
- El valor del `char` asociado al código ASCII puede ser interpretado como `char` o como `int`, mediante lo que se llama un `cast` (método para transformar entre tipos de dato).
- ASCII fue de los primeros códigos para representar texto en un ordenador.
- Se queda corto a la hora de representar idiomas más allá del inglés y algún otro idioma occidental. Idiomas como árabe, japonés, chino, etc. Se representa mejor de otras formas, la más común de ellas es Unicode.
- A diferencia de ASCII, Unicode permite representar muchos idiomas en un ordenador.



ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

8. AUTO

- Tipo de dato que permite al compilador deducir el tipo de dato que se está empleando basándose en el valor con el que inicializamos la variable, es muy útil cuando tienes muchos tipos de datos en un proyecto, o por ejemplo cuando es un tipo de dato muy largo de escribir.

9. ASIGNACIONES

- Una vez una variable es declarada e inicializada, puedo ser asignada de nuevo y adquirir un nuevo valor.
- Hay que tener cuidado al asignar nuevos valores a variables que han sido declaradas e inicializadas con `auto` anteriormente, ya que pueden darse errores, y puede ser un infierno encontrarlos.

10. RESUMEN

- Tipos de dato: `int`, `double`, `float`, `char`, `bool`, `auto`, …
- Los datos se representan en memoria como 0 y 1. Usamos los sistemas numéricos para representar los datos de una forma más accesible para los humanos. Los 3 más importantes son: Binario, Octal y Hexadecimal.