

Tipos Básicos de Dados

# TIPOS CARACTERE E BOOLEANO

# Introdução

- Computadores trabalham com diversos tipos de dados:
  - **Texto** (letras, números, pontuação, etc.)
  - **Números** (naturais, reais, complexos, etc.)
  - **Áudio** (wav, mp3, ogg, etc.)
  - **Imagen** (bmp, jpg, gif, png, tga, etc.)
  - **Vídeo** (avi, mpg, wmv, etc.)
- Todos estes dados são representados pelo computador como um **conjunto de bits**

# Tipos de Dados

- Se distinguem pela natureza dos valores armazenados:
  - **Tipo Inteiro**: números inteiros positivos e negativos.  
Ex.: 30; -20; 0; -1; 390065
  - **Tipo Caractere**: letras, símbolos, números, pontuação.  
Ex.: a, x, k, {, ], !, \$, 3, #
  - **Tipo Ponto Flutuante**: números reais positivos e negativos.  
Ex.: 1.25; -30.54; 0.003;  $2 \times 10^{-8}$
  - **Tipo Booleano**: verdadeiro ou falso.  
Ex.: true, false, 0, 1

# Tipos Inteiros

- Os tipos inteiros da linguagem C++ são:
  - `char` (8 bits)
  - `short int` (16 bits)
  - `int` (32 bits)
  - `long int` (32 bits)
  - `long long int` (64 bits)
- Todos os tipos inteiros são tipos com sinal
  - Representam números **positivos e negativos**

# Tipo Caractere

- O tipo **char** armazena **inteiros de 8 bits**
  - Números de **-128** a **127** para **char**
  - Números de **0** a **255** para **unsigned char**

```
char num = 65;      // char é um tipo inteiro
```

- O tipo **char** é um **tipo inteiro** que é utilizado para armazenar caracteres

```
char letra = 'A';    // caracteres são códigos inteiros
```

# Tipo Caractere

- O computador representa **letras e símbolos** com **números**
  - Um **conjunto de caracteres** é mapeado para uma faixa de números usando uma tabela
    - Existem várias tabelas, como por exemplo:
      - EBCDIC (Mainframes IBM)
      - ASCII (padrão americano)
      - Unicode (suporte internacional)
- A tabela mais tradicional é a **tabela ASCII** (Unicode é compatível com ASCII)

# Tabela ASCII

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr		Dec	Hx	Oct	Html	Chr				
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	Ø	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	\$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Original  
Formada por  
128 caracteres

93	✚	209	〒	225	฿	241	±
94	₩	210	₩	226	₲	242	≥
95	₭	211	₭	227	₮	243	≤
96	₪	212	₪	228	₪	244	₪
97	₩	213	₣	229	₵	245	₵
98	₼	214	₼	230	₼	246	₼
99	₼	215	₼	231	₼	247	₼
00	₼	216	₼	232	₼	248	₼
01	₱	217	₱	233	₵	249	₵
02	₼	218	₼	234	₵	250	₵
03	₼	219	₵	235	₵	251	₵
04	₼	220	₵	236	₵	252	₵
05	=	221	₵	237	₵	253	₵
06	₼	222	₵	238	₵	254	₵
07	₼	223	₵	239	₵	255	₵

# Tabela ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr						
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>Ø</b>	96	60	140	&#96;	<b>‘</b>						
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	<b>!</b>	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>‘a’</b>						
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	<b>”</b>	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>‘b’</b>						
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	<b>#</b>	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>‘c’</b>						
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	<b>\$</b>	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>‘d’</b>						
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	<b>%</b>	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>‘e’</b>						
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	<b>&amp;</b>	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>‘f’</b>						
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	<b>‘</b>	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>‘g’</b>						
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	<b>(</b>	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>‘h’</b>						
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	<b>)</b>	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>‘i’</b>						
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	<b>+</b>	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>‘j’</b>						
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	<b>+</b>	128	Ç	144	É	161	í	177	■■■	193	└	209	╥	225	ß	241	±
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	<b>,</b>	129	ü	145	æ	162	ö	178	■■■	194	┐	210	╥	226	Γ	242	≥
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	<b>-</b>	130	é	146	Æ	163	ú	179	_	195	┤	211	└	227	π	243	≤
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	<b>.</b>	131	â	147	ö	164	ñ	180	_	196	-	212	⠇	228	Σ	244	Γ
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	<b>/</b>	132	ã	148	ö	165	Ñ	181	_	197	+	213	⠄	229	σ	245	J
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	<b>0</b>	133	à	149	ö	166	º	182		198	┤	214	⠄	230	μ	246	÷
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	<b>1</b>	134	ã	150	ü	167	º	183	¶	199	┤	215	+	231	τ	247	≈
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	<b>2</b>	135	§	151	ù	168	ö	184	¶	200	⠄	216	+	232	Φ	248	°
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	<b>3</b>	136	è	152	—	169	—	185		201	⠄	217	⠄	233	⊗	249	.
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	<b>4</b>	137	ë	153	Ö	170	—	186		202	⠄	218	⠄	234	Ω	250	.
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	<b>5</b>	138	è	154	Ü	171	¼	187	¶	203	⠄	219	■	235	δ	251	√
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	<b>6</b>	139	í	156	£	172	¼	188	¶	204	⠄	220	■	236	∞	252	-
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	<b>7</b>	140	í	157	¥	173		189	¶	205	=	221	⠄	237	ϕ	253	²
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	<b>8</b>	141	í	158	—	174	«	190	¶	206	⠄	222	⠄	238	ε	254	■
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	<b>9</b>	142	À	159	ƒ	175	»	191	¶	207	⠄	223	⠄	239	∞	255	
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	<b>:</b>	143	Ã	160	á	176	■■■	192	L	208	⠄	224	⠄	240	≡		
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	<b>:</b>																
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<b>&lt;</b>																
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	<b>=</b>																
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	<b>&gt;</b>																
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	<b>?</b>																

Estendida  
Aproveita os 128  
caracteres livres

128	Ç	144	É	161	í	177	■■■	193	└	209	╥	225	ß	241	±
129	ü	145	æ	162	ö	178	■■■	194	┐	210	╥	226	Γ	242	≥
130	é	146	Æ	163	ú	179		195	┤	211	└	227	π	243	≤
131	â	147	ö	164	ñ	180		196	-	212	⠇	228	Σ	244	Γ
132	ã	148	ö	165	Ñ	181		197	+	213	⠄	229	σ	245	J
133	à	149	ö	166	º	182		198	┤	214	⠄	230	μ	246	÷
134	ã	150	ü	167	º	183	¶	199	┤	215	+	231	τ	247	≈
135	§	151	ù	168	ö	184	¶	200	⠄	216	+	232	Φ	248	°
136	è	152	—	169	—	185		201	⠄	217	⠄	233	⊗	249	.
137	ë	153	Ö	170	—	186		202	⠄	218	⠄	234	Ω	250	.
138	è	154	Ü	171	¼	187	¶	203	⠄	219	■	235	δ	251	√
139	í	156	£	172	¼	188	¶	204	⠄	220	■	236	∞	252	-
140	í	157	¥	173		189	¶	205	=	221	⠄	237	ϕ	253	²
141	í	158	—	174	«	190	¶	206	⠄	222	⠄	238	ε	254	■
142	À	159	ƒ	175	»	191	¶	207	⠄	223	⠄	239	∞	255	
143	Ã	160	á	176	■■■	192	L	208	⠄	224	⠄	240	≡		

# Tipo Caractere

```
// o tipo caractere
#include <iostream>
using namespace std;

int main()
{
    char ch;    // declara uma variável caractere

    cout << "Digite um caractere: " << endl;
    cin >> ch;

    cout << "Olá! ";
    cout << "Obrigado pelo caractere " << ch << '.' << endl;

    return 0;
}
```

# Tipo Caractere

- Saída do programa:

Digite um caractere:

M

Olá! Obrigado pelo caractere M.

- Digitando M o conteúdo da variável ch é o valor 77
- cin e cout fazem as conversões necessárias de inteiro para caractere e vice-versa

# Tipo Caractere

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

int main()
{
    char ch = 'M';      // atribui código ASCII do M
    int i = ch;         // armazena mesmo código num int

    cout << "O Código ASCII para " << ch << ": " << i << endl;
    cout << "Adicionando 1 ao código caractere..." << endl;
    ch = ch + 1;
    i = ch;

    cout << "O Código ASCII para " << ch << ": " << i << endl;
}
```

# Tipo Caractere

- A saída do programa:

O código ASCII para M: 77

Adicionando 1 ao código caractere...

O código ASCII para N: 78

- Como char é um tipo inteiro, pode-se realizar operações matemáticas com os valores armazenados

```
char ch = 'M';    // atribui código ASCII do M  
ch = ch + 1;    // ch = 77 + 1
```

# Constantes Caracteres

- A forma mais simples de representar uma constante caractere é colocar o caractere entre aspas simples

```
char ch = 'M'; // atribui código ASCII do M  
char ch = 77; // mesmo efeito
```

- Recomenda-se utilizar a notação com aspas:
  - É mais clara e direta
  - Não assume uma codificação particular (ASCII)

# Constantes Caracteres

- Alguns caracteres são tratados como **caracteres especiais**:

Caractere	Símbolo ASCII	Código C++
Nova Linha	CR/LF	\n
Tabulação	HT	\t
Backspace	BS	\b
Alerta	BEL	\a
Contra-barra	\	\\
Aspa Simples	'	'
Aspa Dupla	"	\"

# Constantes Caracteres

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

int main()
{
    char alarme = '\a';    // caractere beep
    int senha;

    cout << "Digite a senha: _____\b\b\b\b\b\b\b\b";
    cin >> senha;

    cout << alarme << "Sua senha foi roubada!\a\n";
    cout << "Joãozinho \"O Hacker\" esteve aqui!\n";

    return 0;
}
```

# Constantes Caracteres

- A saída do programa:

```
Digite a sua senha: progcomp
```

```
Sua senha foi roubada!
```

```
Joãozinho "O Hacker"
```

```
esteve aqui!
```

- Alguns compiladores não reconhecem \a (ele pode ser substituído por \007)
- Alguns sistemas podem mostrar \b como um pequeno retângulo ou então apagar os caracteres ao retornar

# Tipo Booleano

- O tipo **bool** armazena um dos valores booleanos
  - Verdadeiro: **true**
  - Falso: **false**

```
bool pronto = false;      // pronto é uma variável booleana
```

- O tipo **bool** ocupa **1 byte** (8 bits) e não 1 bit
  - A CPU não endereça nada menor que 1 byte

```
bool aviso;
cout << sizeof(aviso) << " byte(s)" << endl;    // 1 byte(s)
```

# Tipo Booleano

```
// tipo booleano
#include <iostream>
using namespace std;

int main()
{
    bool buzinar = false;    // buzina desligada

    cout << "Buzinar? ";
    cin >> buzinar;          // leitura de um booleano

    if (buzinar == true)
        cout << "Buzina\alalalalalal";
    else
        cout << "Silêncio" << endl;

    return 0;
}
```

# Tipo Booleano

- A saída do programa:

Buzinar? 1  
Buzina

Buzinar? true  
Silêncio

- O tipo **bool** aceita:

- As constantes **true** e **false** na atribuição
- Qualquer **número** na leitura com cin ou na atribuição
  - Zero é falso
  - Qualquer outro número (positivo ou negativo) é verdadeiro

# Operadores Bit a Bit

- A linguagem C++ oferece um conjunto de operadores para trabalhar com a **representação binária** dos **inteiros**

Operador	Significado	Uso
<code>~</code>	NOT	<code>~expr</code>
<code>&lt;&lt;</code>	LEFT SHIFT	<code>expr1 &lt;&lt; expr2</code>
<code>&gt;&gt;</code>	RIGHT SHIFT	<code>expr1 &gt;&gt; expr2</code>
<code>&amp;</code>	AND	<code>expr1 &amp; expr2</code>
<code> </code>	OR	<code>expr1   expr2</code>
<code>^</code>	XOR	<code>expr1 ^ expr2</code>

# Operadores Bit a Bit

- NOT ( $\sim$ ): inverte todos os bits do operando

```
unsigned char estado = 1;  
estado = ~estado;
```

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1

=

1
---

7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	0

=

254
-----

binário

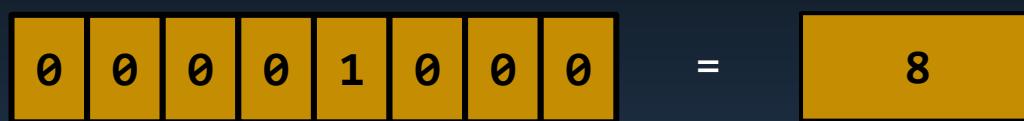
inteiro

A	$\sim A$
0	1
1	0

# Operadores Bit a Bit

- LEFT SHIFT (`<<`): desloca bits para a esquerda

```
unsigned char estado = 1;  
estado = estado << 3;
```



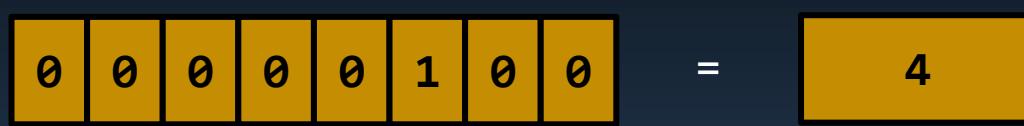
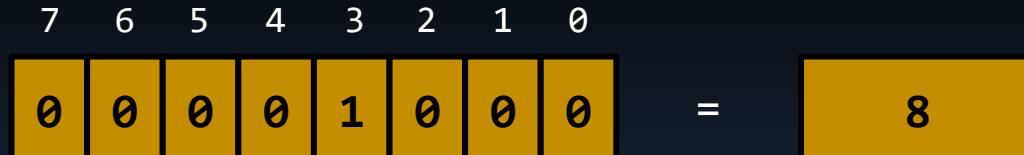
binário

inteiro

# Operadores Bit a Bit

- **RIGHT SHIFT (>>)**: desloca bits para a direita

```
unsigned char estado = 8;  
estado = estado >> 1;
```



binário

inteiro

# Operadores Bit a Bit

- AND (&): faz um AND entre os bits dos seus operandos

```
unsigned char estado = 170;  
estado = estado & 15;
```

7	6	5	4	3	2	1	0		=	inteto
1	0	1	0	1	0	1	0		=	170
&										
0	0	0	0	1	1	1	1		=	15

A	B	A & B
0	0	0
1	0	0
0	1	0
1	1	1

# Operadores Bit a Bit

- OR (): faz um OR entre os bits dos seus operandos

```
unsigned char estado = 170;  
estado = estado | 15;
```

7	6	5	4	3	2	1	0	
1	0	1	0	1	0	1	0	= 170
0	0	0	0	1	1	1	1	= 15
1	0	1	0	1	1	1	1	= 175

A	B	A   B
0	0	0
1	0	1
0	1	1
1	1	1

# Operadores Bit a Bit

- XOR (^): faz um XOR bit a bit entre seus operandos

```
unsigned char estado = 170;  
estado = estado ^ 15;
```

7	6	5	4	3	2	1	0	
1	0	1	0	1	0	1	0	= 170
0	0	0	0	1	1	1	1	= 15
<hr/>								
1	0	1	0	0	1	0	1	= 165
binário								inteiro

A	B	$A \wedge B$
0	0	0
1	0	1
0	1	1
1	1	0

# Operações com Bits

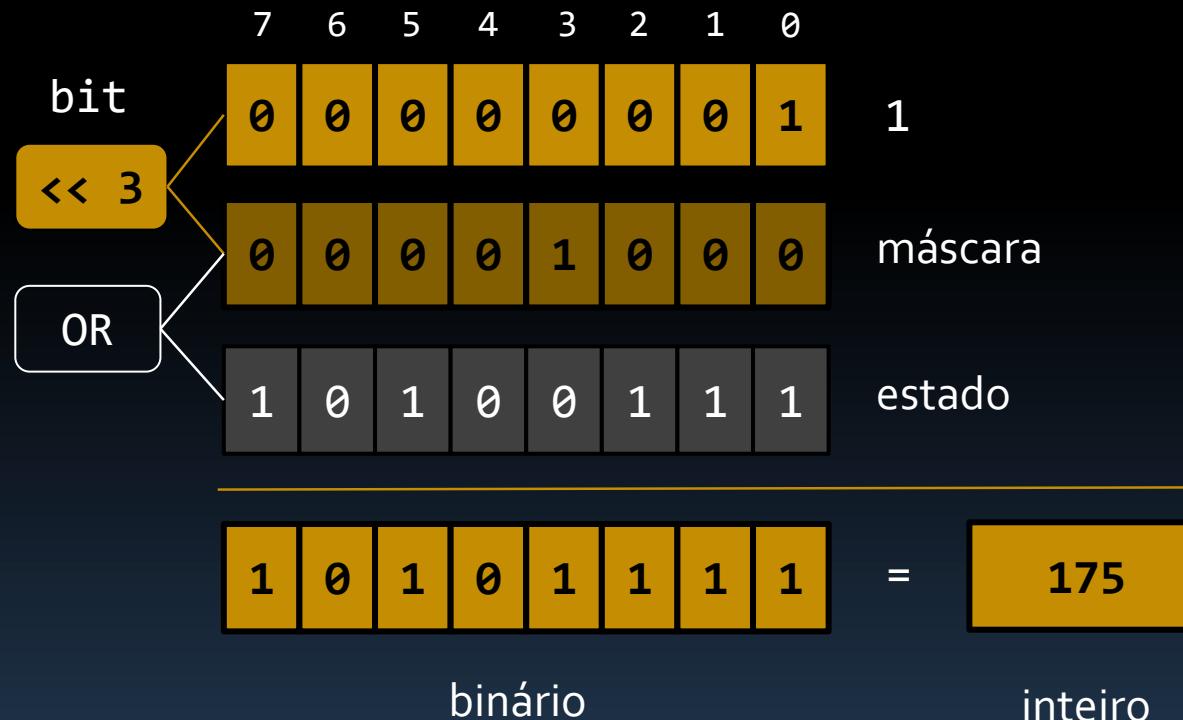
- Ligando um bit:

```
cout << "Ligar qual bit? ";
int bit;
cin >> bit;
unsigned char mascara = 1 << bit;

unsigned char estado = 167;
estado = estado | mascara;
cout << int(estado) << endl;
```

- Saída:

Ligar qual bit? 3  
175



# Operações com Bits

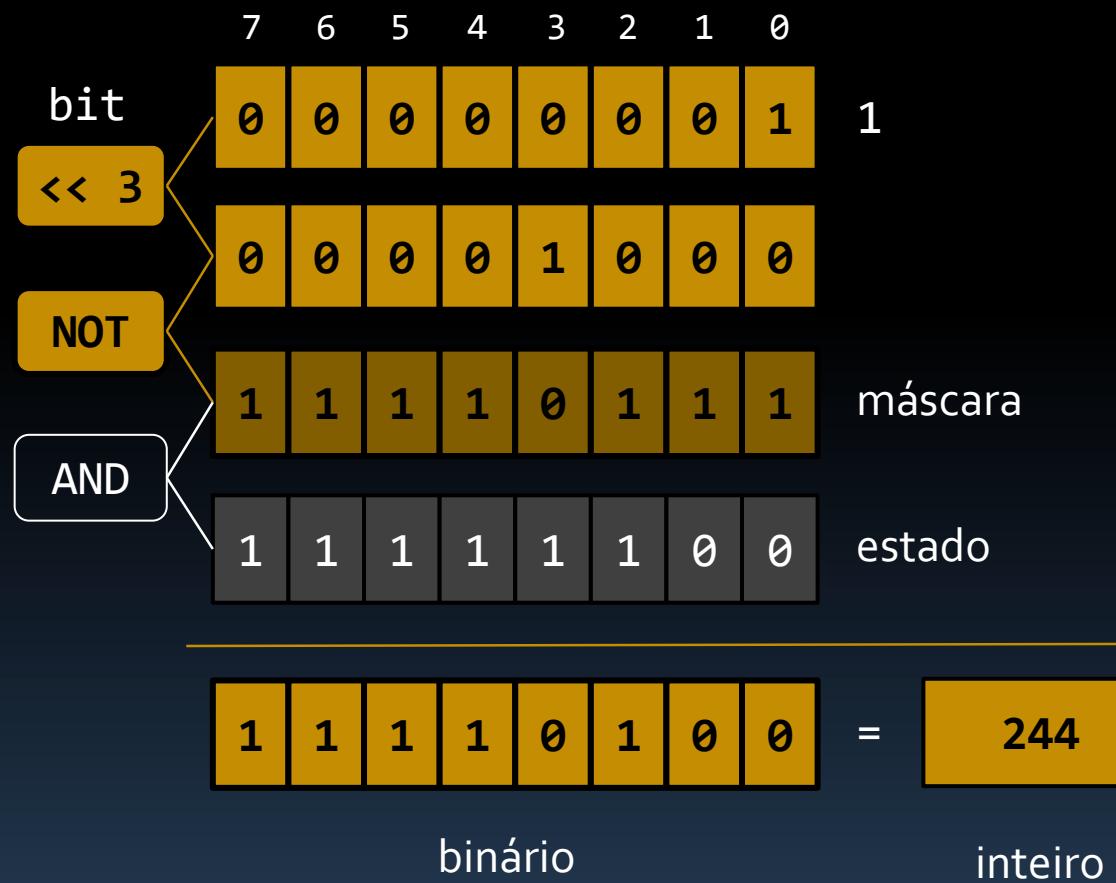
- Desligando um bit:

```
cout << "Desligar qual bit? ";
int bit;
cin >> bit;
unsigned char mascara = ~(1 << bit);

unsigned char estado = 252;
estado = estado & mascara;
cout << int(estado) << endl;
```

- Saída:

Desligar qual bit? 3  
244



# Operações com Bits

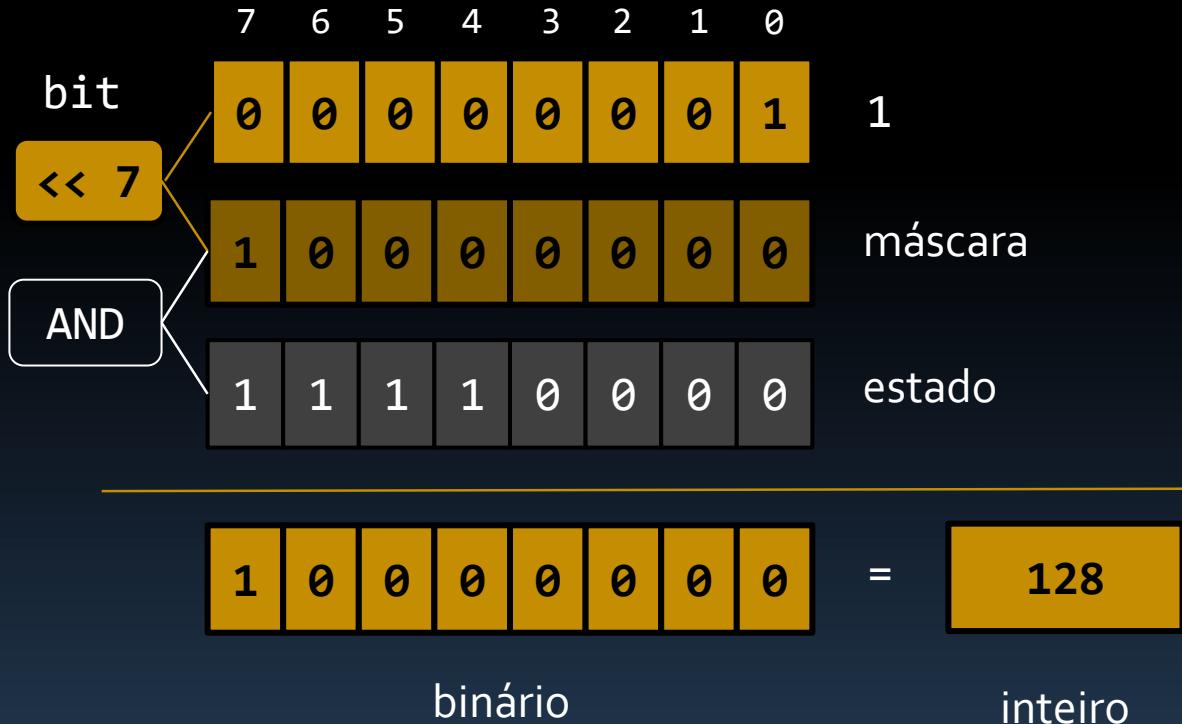
- Testando um bit:

```
cout << "Testar qual bit? ";
int bit;
cin >> bit;
unsigned char mascara = 1 << bit;

unsigned char estado = 240;
if (estado & mascara)
    cout << "ligado" << endl;
else
    cout << "desligado" << endl;
```

- Saída:

Testar qual bit? 7  
ligado



# Resumo

Tipos Inteiros	Bits	Faixa
bool	8	0 a 1
char	8	-128 a 127
unsigned char	8	0 a 255
short	16	-32.768 a 32.767
unsigned short	16	0 a 65.535
int	32	-2.147.483.648 a 2.147.483.647
unsigned int	32	0 a 4.294.967.295
long	32	-2.147.483.648 a 2.147.483.647
unsigned long	32	0 a 4.294.967.295
long long	64	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
unsigned long long	64	0 a 18.446.744.073.709.661.615

# Resumo

- O tipo **char** é usado para **representar caracteres** usando uma codificação numérica estabelecida pela **tabela ASCII**
  - Ele guarda um número inteiro
    - Por isso é possível usar **operações matemáticas**

```
char ch = 'M';    // atribui código ASCII do M  
ch = ch + 1;     // ch = 77 + 1
```

- A linguagem C++ define **caracteres especiais**
  - Utilizam a barra invertida: '\n', '\b', '\t', '\a', etc.

# Resumo

- O tipo **bool** é usado para representar os valores:
  - Verdadeiro (**true**)
  - Falso (**false**)
  - O tipo booleano ocupa 1 byte e não 1 bit
- Os operadores **bit a bit** ~ << >> & | ^
  - Podem ser usados para manipular os bits de valores inteiros
    - Recomenda-se utilizar valores **unsigned** (sem sinal)
    - O tipo **unsigned char** pode guardar até 8 booleanos de 1 bit