Tipos Básicos de Dados

# TIPOS CARACTERE E BOOLEANO

Judson Santos Santiago

# Introdução

- Computadores trabalham com diversos tipos de dados:
  - Texto (letras, números, pontuação, etc.)
  - Números (naturais, reais, complexos, etc.)
  - Áudio (wav, mp3, ogg, etc.)
  - Imagem (bmp, jpg, gif, png, tga, etc.)
  - Vídeo (avi, mpg, wmv, etc.)

- Estes dados se classificam em tipos de dados básicos e tipos de dados compostos

# Tipos de Dados

- Os tipos básicos de dados se distinguem pela natureza dos valores armazenados:
  - Tipo Inteiro: números naturais positivos e negativos.
    Ex.: `30; -20; 0; -1; 390065`
  - Tipo Caractere: letras, símbolos, números, pontuação.
    Ex.: `a, x, k, {, ], !, $, 3, #`
  - Tipo Booleano: verdadeiro ou falso.
    Ex.: `true, false, 0, 1`
  - Tipo Ponto Flutuante: números reais positivos e negativos.
    Ex.: `1.25; -30.54; 0.003;` $2x10^{-8}$

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

## Original

Formada por 128 caracteres

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 143 | Å | 160 | á | 176 | ▒ | 192 | └ | 208 | ╨ | 224 | α | 240 | ≡ | | | | | |

# Tabela ASCII

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | | | | | | | | | | |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | | | | | | | | | | |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | | | | | | | | | | |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | | | | | | | | | | |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | | | | | | | | | | |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | | | | | | | | | | |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | | | | | | | | | | |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | | | | | | | | | | |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | | | | | | | | | | |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | | | | | | | | | | |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | | | | | | | | | | |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | | | | | | | | | | |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | | | | | | | | | | |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | | | | | | | | | | |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | | | | | | | | | | |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | | | | | | | | | | |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | | | | | | | | | | |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | | | | | | | | | | |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | | | | | | | | | | |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | | | | | | | | | | |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | | | | | | | | | | |

## Estendida

Aproveita os 128 caracteres livres

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 128 | Ç | 144 | É | 161 | í | 177 | ▒ | 193 | ┴ | 209 | ╤ | 225 | ß | 241 | ± |
| 129 | ü | 145 | æ | 162 | ó | 178 | ▓ | 194 | ┬ | 210 | ╥ | 226 | Γ | 242 | ≥ |
| 130 | é | 146 | Æ | 163 | ú | 179 | │ | 195 | ├ | 211 | ╙ | 227 | π | 243 | ≤ |
| 131 | â | 147 | ô | 164 | ñ | 180 | ┤ | 196 | ─ | 212 | ╘ | 228 | Σ | 244 | ⌠ |
| 132 | ä | 148 | ö | 165 | Ñ | 181 | ╡ | 197 | ┼ | 213 | ╒ | 229 | σ | 245 | ⌡ |
| 133 | à | 149 | ò | 166 | ª | 182 | ╢ | 198 | ╞ | 214 | ╓ | 230 | µ | 246 | ÷ |
| 134 | å | 150 | û | 167 | º | 183 | ╖ | 199 | ╟ | 215 | ╫ | 231 | τ | 247 | ≈ |
| 135 | ç | 151 | ù | 168 | ¿ | 184 | ╕ | 200 | ╚ | 216 | ╪ | 232 | Φ | 248 | ° |
| 136 | ê | 152 | _ | 169 | _ | 185 | ╣ | 201 | ╔ | 217 | ┘ | 233 | Θ | 249 | · |
| 137 | ë | 153 | Ö | 170 | ¬ | 186 | ║ | 202 | ╩ | 218 | ┌ | 234 | Ω | 250 | · |
| 138 | è | 154 | Ü | 171 | ½ | 187 | ╗ | 203 | ╦ | 219 | ▉ | 235 | δ | 251 | √ |
| 139 | ï | 156 | £ | 172 | ¼ | 188 | ╝ | 204 | ╠ | 220 | ▄ | 236 | ∞ | 252 | _ |
| 140 | î | 157 | ¥ | 173 | ¡ | 189 | ╜ | 205 | ═ | 221 | ▌ | 237 | ϕ | 253 | ² |
| 141 | ì | 158 | _ | 174 | « | 190 | ╛ | 206 | ╬ | 222 | ▐ | 238 | ε | 254 | ■ |
| 142 | Ä | 159 | ƒ | 175 | » | 191 | ┐ | 207 | ╧ | 223 | ▀ | 239 | ∩ | 255 | |
| 143 | Å | 160 | á | 176 | ▒ | 192 | └ | 208 | ╨ | 224 | α | 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.
```

- Se o usuário digitar a letra M o conteúdo da variável ch é o valor inteiro 77
- cin e cout fazem as conversões necessárias de inteiro para caractere e vice-versa

# Tipo Caractere

```
// o tipo caractere e o tipo inteiro
#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

```
// caracteres especiais
#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\"\nesteve 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

```cpp
// 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\a\a\a\a\a";
    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

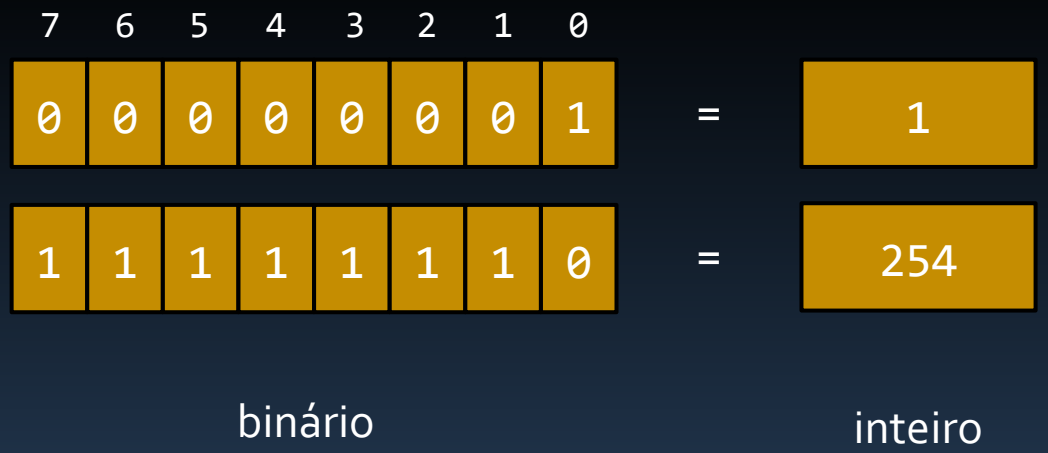
- A linguagem C++ oferece um conjunto de operadores para trabalhar com a representação binária dos inteiros

| Operador | Significado | Uso |
|---|---|---|
| ~ | NOT | ~expr |
| << | LEFT SHIFT | expr1 << expr2 |
| >> | RIGHT SHIFT | expr1 >> expr2 |
| & | AND | expr1 & expr2 |
| \| | OR | expr1 \| expr2 |
| ^ | XOR | expr1 ^ expr2 |

# Operadores Bit a Bit

- NOT (~): inverte todos os bits do operando

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




| A | ~A |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Operadores Bit a Bit

- LEFT SHIFT (<<): desloca uma certa quantidade de bits para a esquerda

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

# Operadores Bit a Bit

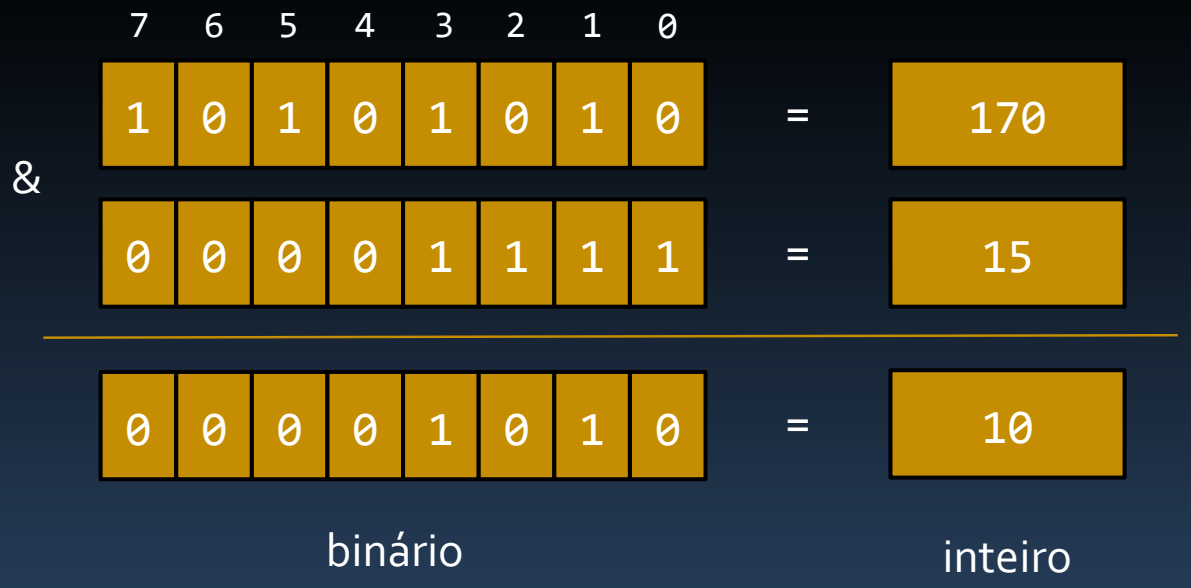
- RIGHT SHIFT (>>): desloca uma certa quantidade de bits para a direita

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

# Operadores Bit a Bit

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

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

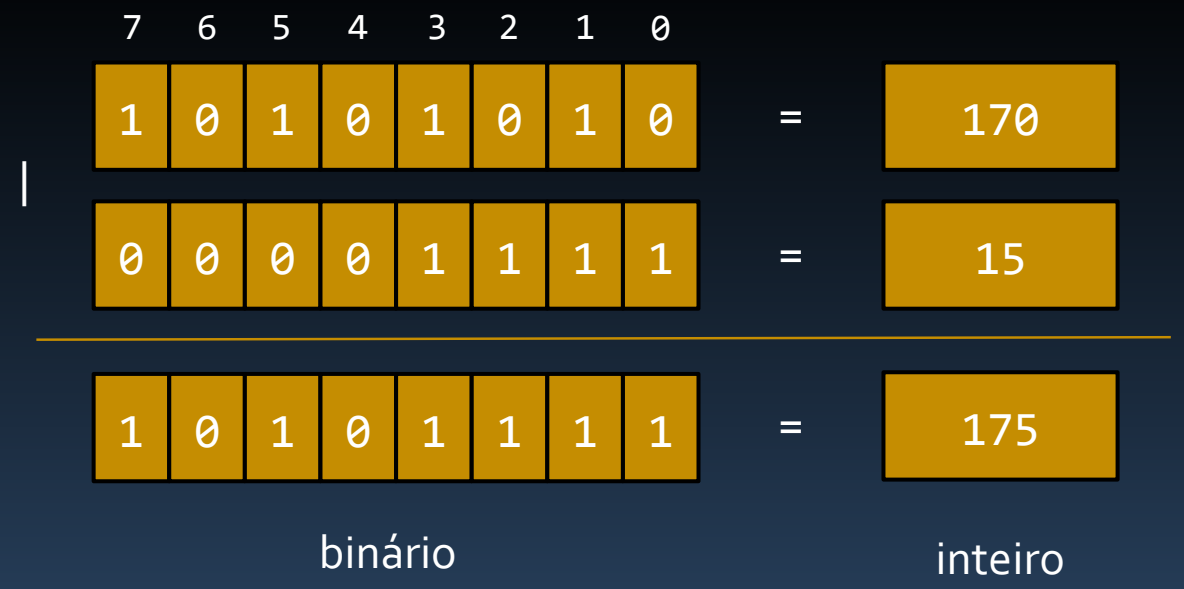



| A | B | A & B |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

# Operadores Bit a Bit

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

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

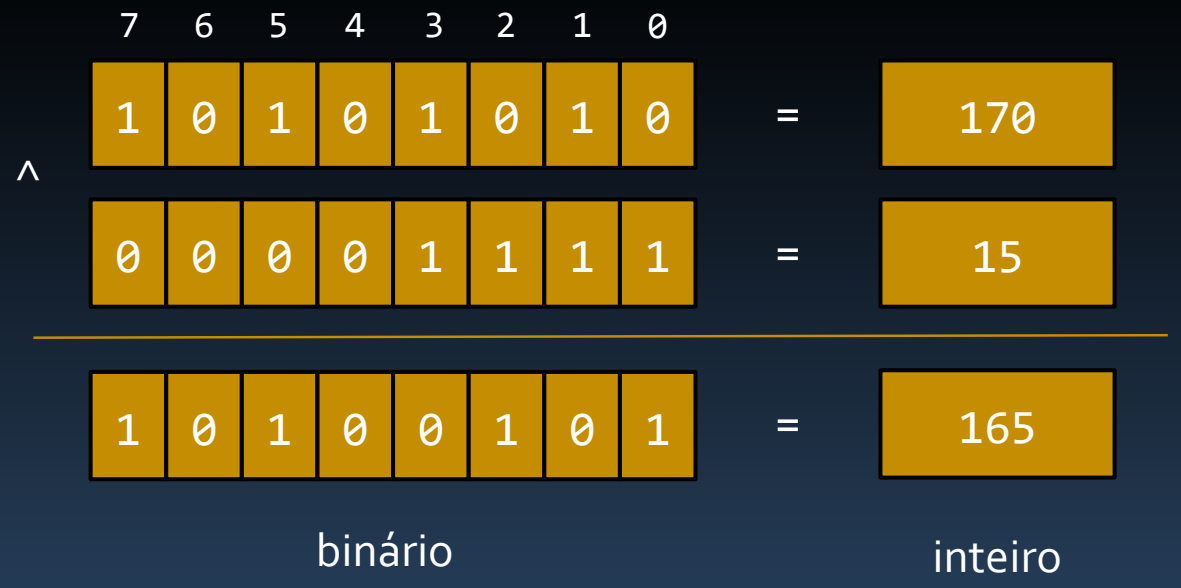



| A | B | A \| B |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

# Operadores Bit a Bit

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

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




| A | B | A ^ 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

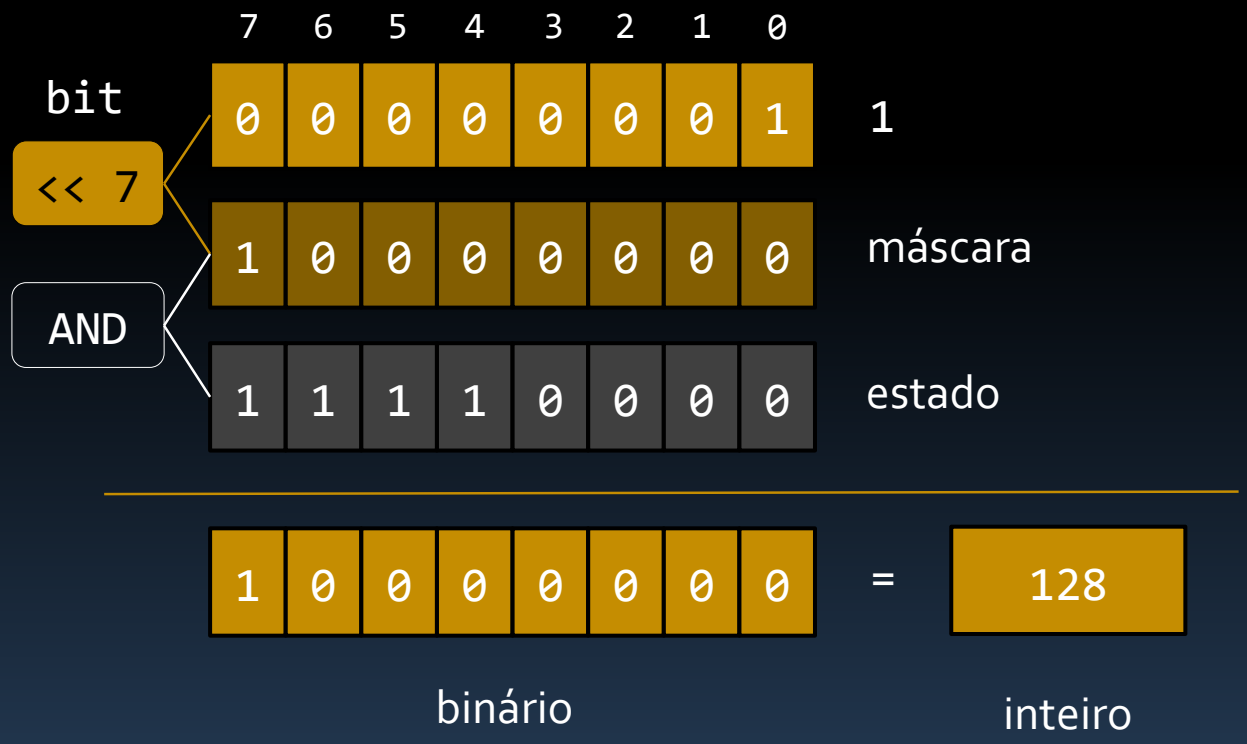
- Testando um bit:

```
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