

Tipos Básicos de Dados

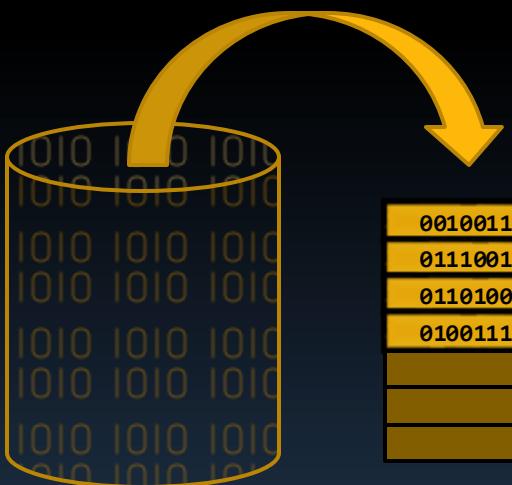
TIPOS INTEIROS

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**

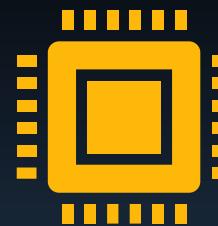
Introdução

- Os dados são gravados em unidades de armazenamento e carregados na memória para execução



HDD / SSD

Memória



A CPU só trabalha
com dados que estão
na memória

Introdução

- Os **programas** obtêm dados:
 - Lendo-os da unidade de armazenamento
 - Transferindo-os para a memória
 - Lendo-os da entrada (teclado) para a memória

- Os **programas** geram dados:
 - Armazenando-os na memória
 - Transferindo-os da memória para arquivos
 - Escrevendo-os na saída (tela)

opcional

opcional

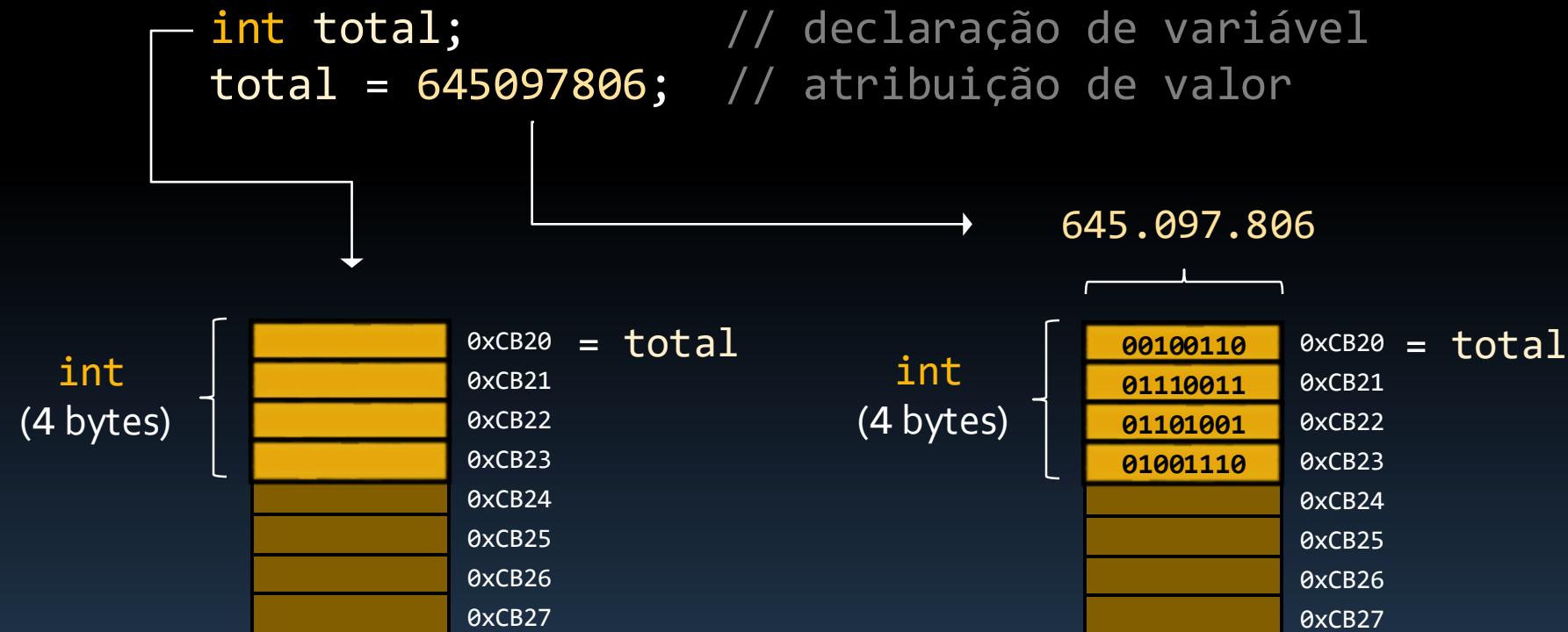
Introdução

- Para guardar dados na memória um programa precisa definir:
 1. Onde os dados serão guardados
 2. Que tipo de dado será armazenado
 3. O valor a ser armazenado



Introdução

- A **criação de variáveis** fornece as informações necessárias



Introdução

- Toda variável precisa de:
 - Um nome: um rótulo para uma posição de memória
 - Um tipo: define o que pode ser guardado naquela variável
 - Um valor: é fornecido através da atribuição de valor

Tipo	Nome	
<code>int total;</code>		// declaração de variável
<code>total = 638334552;</code>		// atribuição de valor
		Valor

Nomes de Variáveis

- C++ impõe as seguinte regras:
 - Os caracteres válidos são caracteres do alfabeto, dígitos numéricos e o caractere sublinhado

abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789

-

- O primeiro caractere de um nome não pode ser um dígito numérico

```
int 4ever; X  
int 7a1; X
```

Nomes de Variáveis

- C++ impõe as seguinte regras:
 - Caracteres **maiúsculos** são diferentes de caracteres **minúsculos**

```
int total;      // variável válida
int Total;      // variável diferente
```

- Uma **palavra-chave** não pode ser usada

```
int using;  ×
int return; ×
```

Nomes de Variáveis

- C++ impõe as seguinte regras:
 - Nomes **iniciando com um ou dois sublinhados** são reservados para a implementação do compilador

```
int _tamanho; // reservado, deve-se evitar
int __inicio; // reservado, deve-se evitar
```
 - Usando um nome de variável como **_tamanho** ou **__inicio** não constitui um erro, mas o comportamento do compilador é indefinido

Nomes de Variáveis

- Em C++ não existe limite para o tamanho de um nome

```
// nome grandes são permitidos  
int custoDeManutencaoDoSistemaPorHoradeFuncionamento;
```

- A linguagem C reconhece apenas os 63 primeiros caracteres
- A linguagem C++ estimula a utilização de nomes significativos
 - Use custoDaViagem
 - No lugar de x ou custo

Nomes de Variáveis

- Existem vários **estilos** para nomear variáveis
 - Usar **iniciais maiúsculas** ou **sublinhado** é o mais comum

```
int total_geral; // estilo antigo usado em C  
int totalGeral; // tendência mais moderna
```

- Alguns estilos usam um **prefixo** para indicar o tipo da variável (**Hungarian Notation**)

```
int xwMax;      // xw indica posição x da window  
int gMax;       // g indica uma variável global  
int iMax;       // i indica um número inteiro
```

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; 2×10^{-8}
 - **Tipo Booleano**: verdadeiro ou falso.
Ex.: true, false, 0, 1

Tipos Inteiros

- Inteiros são números sem parte fracionária
 - Ex.: 2, 98, -5286, 0
- Existem infinitos números inteiros
 - Nenhuma memória é capaz de representar todos os inteiros
 - Uma linguagem representa apenas um subconjunto destes números
- C++ oferece vários tipos inteiros
 - Eles diferem pela quantidade de memória utilizada

Tipos Inteiros

- A unidade fundamental do computador é o bit
 - Representa os valores 0 ou 1

 0 0 0 0 0 0 0 1

8 bits
representam
256 valores

1 bit

0 - 0
1 - 1

2 bits

00 - 0
01 - 1
10 - 2
11 - 3

3 bits

000 - 0
001 - 1
010 - 2
011 - 3
100 - 4
101 - 5
110 - 6
111 - 7

Cada bit adicional duplica a capacidade de armazenamento.

Tipos Inteiros

- Um **byte** corresponde a 8 bits de memória



A horizontal sequence of eight yellow rectangular boxes, each containing a black zero ('0') or one ('1'). The boxes are arranged side-by-side, representing a single byte of memory.

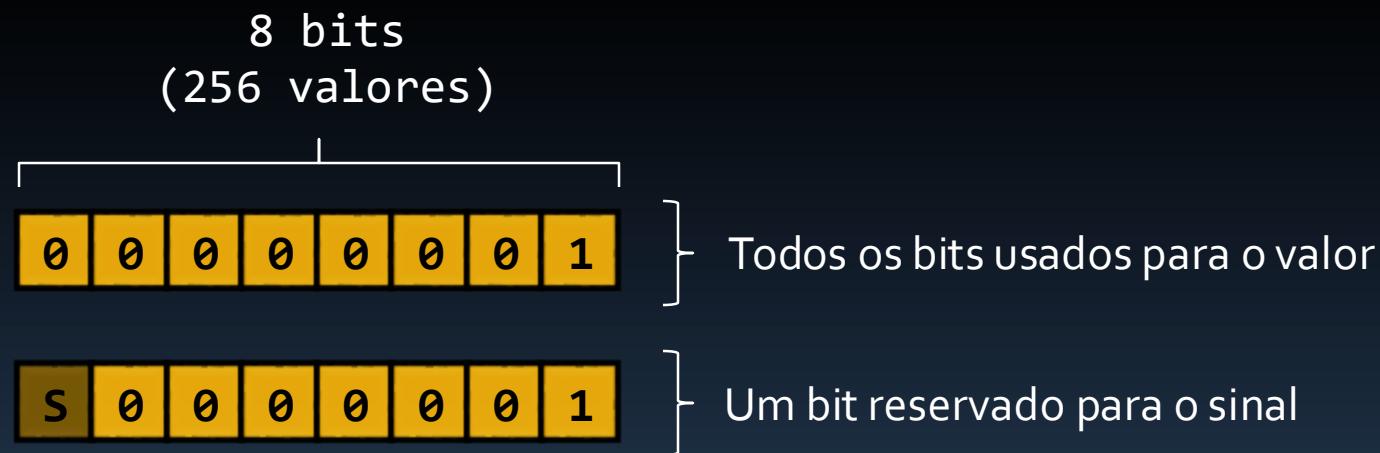
0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

1 byte = 8 bits

- 8 bits (1 byte) representam 256 valores
- 16 bits (2 bytes) representam 65.536 valores
- 32 bits (4 bytes) representam 4.294.672.296 valores
- 64 bits (8 bytes) representam 18.446.744.073.709.661.615 valores

Tipos Inteiros

- Um conjunto de bits pode representar números:
 - Apenas **positivos** ou
 - Positivos e **negativos**



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, ou seja, podem representar números **positivos e negativos**

Tipos Inteiros

- Os tipos derivados do int podem ser encurtados
 - short int
 - int
 - long int
 - long long int

Dificilmente se encontra a forma longa em códigos C++ mas elas podem ser usadas.

```
short placar;           // variável de tipo short int
int temperatura;        // variável de tipo int
long posicao;           // variável de tipo long int
long long populacao;    // variável de tipo long long int
```

Tipos Inteiros

- Não existe uma padronização da quantidade de bits que os tipos inteiros devem usar em cada plataforma
 - O padrão da linguagem C++ estipula apenas tamanhos mínimos:
 - short é pelo menos 16 bits
 - int é pelo menos tão largo quanto short
 - long é pelo menos 32 bits e tão largo quanto int
 - long long é pelo menos 64 bits e tão largo quanto long

Tipos Inteiros

- A maioria dos sistemas usam os valores mínimos para:
`short (16 bits)` e `long long (64 bits)`
- Porém existem várias possibilidades para o `int`:
 - `int` tem 16 bits (o mesmo que `short`) para as implementações antigas do PC: DOS, Windows 3.11, OS2
 - `int` tem 32 bits (o mesmo que `long`) para as implementações mais recentes: Windows 11, Linux, MacOS

Tipos Inteiros

- Existem várias possibilidades para o **long**:
 - **long** tem 32 bits (o mesmo que **int**):
 - Windows (32 bits e 64 bits)
 - Linux (32 bits), MacOS (32 bits)
 - **long** tem 64 bits (o mesmo que **long long**):
 - Linux (64 bits), MacOS (64 bits)
- Existem sobreposições por razões históricas
 - Tentativa de manter compatibilidade entre sistemas

Tipos Inteiros

- Para saber o tamanho dos tipos inteiros em uma determinada plataforma pode-se utilizar:
 - O operador `sizeof`
 - Ele retorna o número de bytes ocupados por um tipo ou uma variável
 - O arquivo de cabeçalho `climits`
 - Contém informações sobre os limites dos tipos inteiros:
 - `INT_MAX`: o maior valor `int` possível
 - `INT_MIN`: o menor valor `int` possível

Tipos Inteiros

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

int main()
{
    int n_int = INT_MAX;
    short n_short = SHRT_MAX;
    long n_long = LONG_MAX;
    long long n_llong = LLONG_MAX;

    cout << "short tem " << sizeof n_short << " bytes." << endl;
    cout << "int tem " << sizeof(int) << " bytes." << endl;
    cout << "long tem " << sizeof n_long << " bytes." << endl;
    cout << "long long tem " << sizeof n_llong << " bytes.\n" << endl;

    cout << "Valores Máximos:\n";
    cout << "short: " << n_short << endl;
    cout << "int: " << n_int << endl;
    cout << "long: " << n_long << endl;
    cout << "long long: " << n_llong << endl << endl;
}
```

Tipos Inteiros

- A saída do programa usando o compilador Borland C++ no MS-DOS:

```
short tem 2 bytes.  
int tem 2 bytes.  
long tem 4 bytes.  
long long tem 8 bytes.
```

Valores Máximos:

```
short: 32767  
int: 32767  
long: 2147483647  
long long: 9223372036854775807
```

Tipos Inteiros

- A saída do programa usando o compilador Visual C++ no Windows 11:

```
short tem 2 bytes.  
int tem 4 bytes.  
long tem 4 bytes.  
long long tem 8 bytes.
```

Valores Máximos:

```
short: 32767  
int: 2147483647  
long: 2147483647  
long long: 9223372036854775807
```

Tipos Inteiros

- A saída do programa usando o compilador **g++** no Linux:

```
short tem 2 bytes.  
int tem 4 bytes.  
long tem 8 bytes.  
long long tem 8 bytes.
```

Valores Máximos:

```
short: 32767  
int: 2147483647  
long: 9223372036854775807  
long long: 9223372036854775807
```

O Operador sizeof

- O operador **sizeof** pode trabalhar tanto com **tipos** quanto com **variáveis**

```
cout << "int tem " << sizeof(int) << " bytes." << endl;  
cout << "short tem " << sizeof n_short << " bytes." << endl;
```

- Os parênteses são obrigatórios quando o operando é um tipo
- Os parênteses são opcionais quando o operando é um nome de variável

O Arquivo `climits`

- O arquivo de cabeçalho `climits` define **constantes simbólicas**

Constante	Representa
<code>SHRT_MAX</code>	Valor máximo para <code>short</code>
<code>SHRT_MIN</code>	Valor mínimo para <code>short</code>
<code>INT_MAX</code>	Valor máximo para <code>int</code>
<code>INT_MIN</code>	Valor mínimo para <code>int</code>
<code>LONG_MAX</code>	Valor máximo para <code>long</code>
<code>LONG_MIN</code>	Valor mínimo para <code>long</code>
<code>LLONG_MIN</code>	Valor mínimo para <code>long long</code>
<code>LLONG_MAX</code>	Valor máximo para <code>long long</code>

Constantes Simbólicas

- O arquivo **climits** contém linhas como esta:

```
#define INT_MAX 2147483647  
#define INT_MIN -2147483648
```

- **#define** é uma outra diretiva de pré-processamento
 - Ela funciona como um "**localizar e substituir**"
 - O **pré-processador** procura pelas ocorrências de **INT_MAX** no código fonte e substitui por **21476483647**

Constantes Simbólicas

```
// horasseg.cpp - converte horas em segundos
#include <iostream>
using namespace std;

#define SEGPORHORA 3600

int main()
{
    cout << "Digite uma quantidade de tempo em horas: ";
    int horas;
    cin >> horas;

    int segundos = horas * SEGPORHORA;
    cout << "Isso equivale a " << segundos << " segundos.\n";
}
```

Inicialização de Variáveis

- A inicialização **combina** a declaração com a atribuição de **valor** em uma única instrução

```
int n = INT_MAX;
```

- A inicialização pode ser feita com **constantes, variáveis e expressões**

```
int alunos = 5;
int alunas = alunos;
int cadeiras = alunos + alunas + 4 + converte(10);
```

Inicialização de Variáveis

- C++ possui **formas alternativas para inicializar** variáveis que não são válidas em C

```
int num = 101;      // válido em C/C++
int num(101);      // válido somente em C++
int num = {101};    // válido a partir do C++11
int num{101};       // válido a partir do C++11
```

- Uma **variável local não inicializada** contém um valor indefinido (lixo da memória)

```
short ano;        // valor indefinido
cout << ano;      // mostrará lixo da memória
```

Tipos Sem Sinal

- Cada um dos tipos inteiros possui uma versão sem sinal que guarda apenas números positivos

```
unsigned short tam;          // tipo unsigned short  
unsigned int mesas;         // tipo unsigned int  
unsigned alunos;            // mesmo que unsigned int  
unsigned long cadeiras;     // tipo unsigned long  
unsigned long long pop;     // tipo unsigned long long
```

- Eliminando os negativos sobra mais espaço para os positivos:
 - short representa números de -32768 a 32767
 - unsigned short representa números de 0 a 65535

Overflow e Underflow

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

int main()
{
    short pedro = SHRT_MAX;
    unsigned short maria = SHRT_MAX;

    cout << "Pedro tem " << pedro << " Reais." << endl;
    cout << "Maria tem " << maria << " Reais." << endl;

    cout << endl << "Adicionando 1 Real para cada um..." << endl << endl;
    pedro = pedro + 1;
    maria = maria + 1;

    cout << "Agora Pedro tem " << pedro << " Reais." << endl;
    cout << "Agora Maria tem " << maria << " Reais." << endl;
}
```

Overflow e Underflow

- A saída do programa:

Pedro tem 32767 Reais.

Maria tem 32767 Reais.

Adicionando 1 Real para cada um...

Agora Pedro tem -32768 Reais.

Agora Maria tem 32768 Reais.

Overflow e Underflow

```
#include <iostream>
#include <climits>
#define ZERO 0
using namespace std;
int main()
{
    short pedro = ZERO;
    unsigned short maria = ZERO;

    cout << "Pedro tem " << pedro << " Reais." << endl;
    cout << "Maria tem " << maria << " Reais." << endl;

    cout << endl << "Tirando 1 Real de cada um..." << endl << endl;
    pedro = pedro - 1;
    maria = maria - 1;

    cout << "Agora Pedro tem " << pedro << " Reais." << endl;
    cout << "Agora Maria tem " << maria << " Reais." << endl;
}
```

Overflow e Underflow

- A saída do programa:

Pedro tem 0 Reais.

Maria tem 0 Reais.

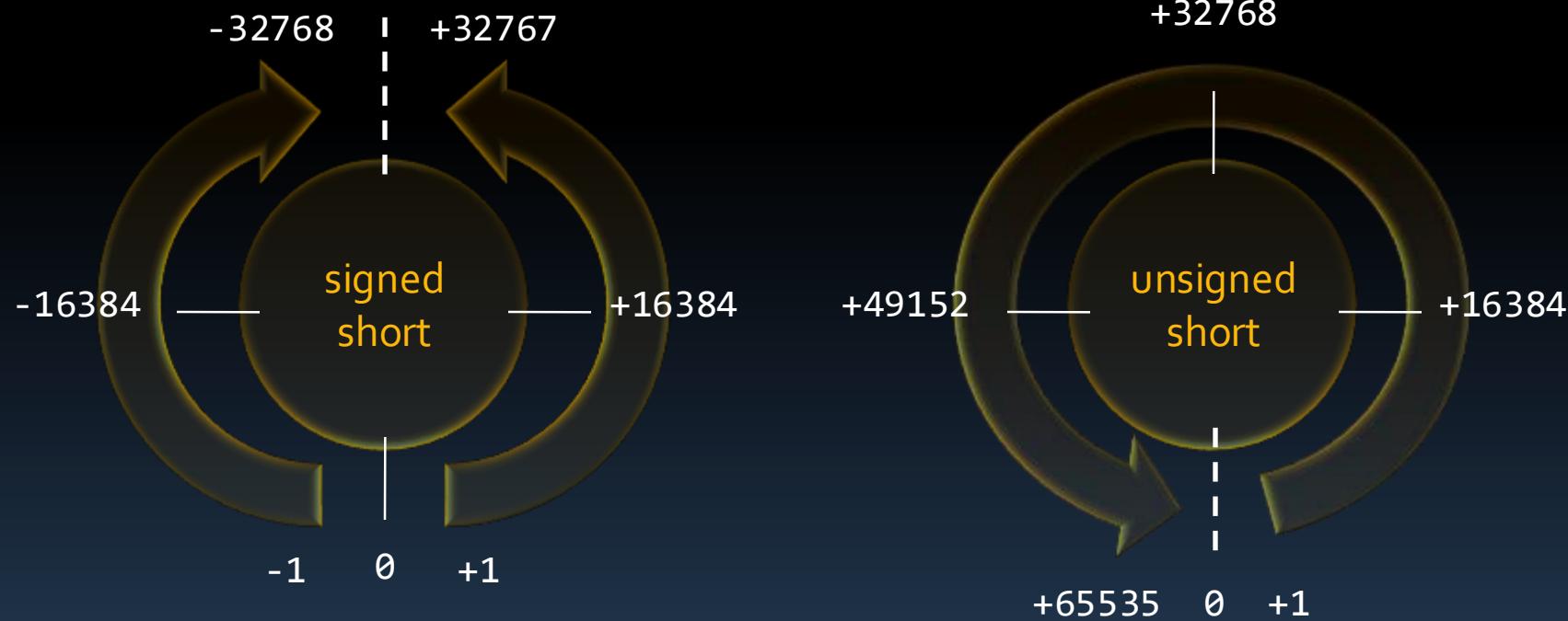
Tirando 1 Real de cada um...

Agora Pedro tem -1 Reais.

Agora Maria tem 65535 Reais.

Overflow e Underflow

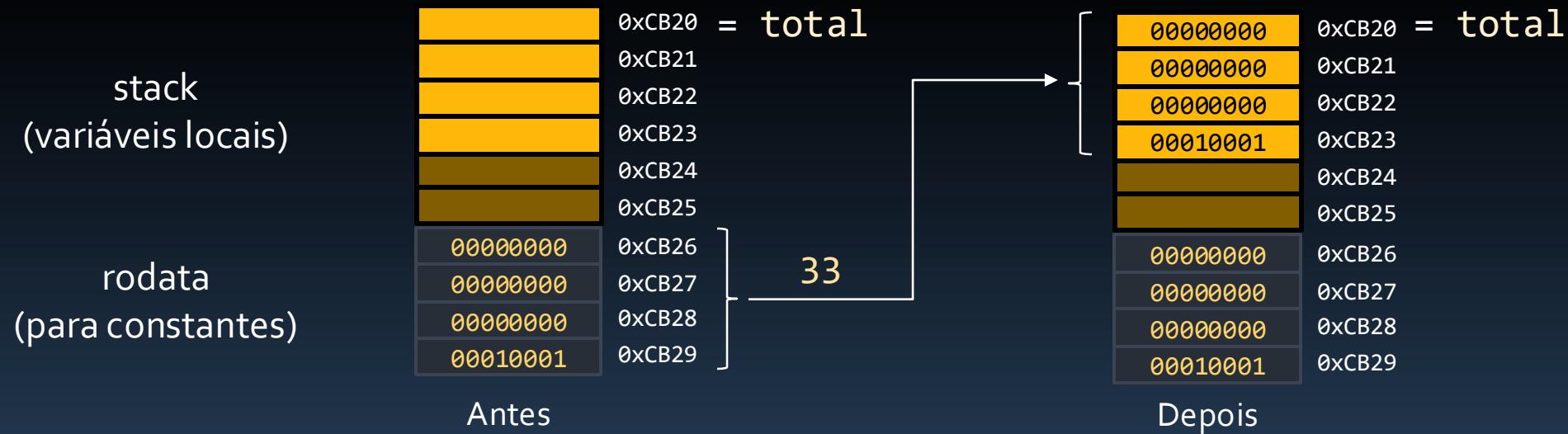
- Os tipos inteiros se comportam como um **odômetro analógico**



Constantes Inteiras

- Uma constante também ocupa espaço na memória

```
int total;      // declaração de variável  
total = 33;    // atribuição de valor
```



Constantes Inteiras

- É possível indicar a quantidade de espaço a ser usado

```
int total = 33;          // constante int
int total = 33L;         // constante long
int total = 33UL;        // constante unsigned long
int total = 33LL;        // constante long long
```



Constantes Inteiras

- Por que o programa abaixo não exibe o resultado correto?

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

int main()
{
    int total = 100000000 + 200000000;
    int multi = 100000000 * 200000000;

    cout << "Total: " << total << endl;
    cout << "Multi: " << multi << endl;

    return 0;
}
```

Qual Tipo Inteiro Escolher?

- C++ possui uma grande variedade de tipos disponíveis:
 - `int` normalmente oferece o melhor desempenho
 - Use `unsigned` para valores que nunca podem ser negativos
 - Se compatibilidade com plataformas antigas é importante, utilize `long` para representar valores maiores que 32767
 - Utilize `short` apenas para conservar memória, especialmente em um vetor de inteiros
 - Se 2 bilhões é pouco, use `long long`

Qual Tipo Inteiro Escolher?

Tipos Inteiros	Bits	Faixa
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

Tabela válida para o compilador do Visual Studio no Windows (x86 e x64)

Resumo

- A linguagem C++ conta com diversos **tipos de dados para representar valores inteiros**:
 - short (**16 bits**)
 - int (**16, 24 ou 32 bits**)
 - long (**32 bits ou 64 bits**)
 - long long (**64 bits**)
- Cada um destes tipos possui uma versão com sinal (**signed**) e outra sem sinal (**unsigned**)