Operadores

Programação e Desenvolvimento de Software I

Gleison S. D. Mendonça, Luigi D. C. Soares
{gleison.mendonca, luigi.domenico}@dcc.ufmg.br

Operadores

- O que é/para que serve um operador?

Operadores

- O que é/para que serve um operador?
- Qual a diferença entre um operador **aritmético** e um operador **relacional**?

Operadores

- O que é/para que serve um operador?
- Qual a diferença entre um operador **aritmético** e um operador **relacional**?
- E o que é um operador **unário**?

Operadores

- O que é/para que serve um operador?
- Qual a diferença entre um operador **aritmético** e um operador **relacional**?
- E o que é um operador **unário**?
- E um operador **binário**?

Operadores

- O que é/para que serve um operador?
- Qual a diferença entre um operador **aritmético** e um operador **relacional**?
- E o que é um operador **unário**?
- E um operador **binário**?
- E um operador **bit a bit**?

Operadores Aritméticos - Binários

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    int y = 3;
    // +: Adição de dois números
    printf("x + y = %d\n", x + y);
    // -: Subtração de dois números
    printf("x - y = %d\n", x - y);
    return 0;
}
```

```
x + y = 13
x - y = 7
```

Operadores Aritméticos - Binários

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    int y = 3;
    // *: Multiplicação de dois números
    printf("x * y = %d\n", x * y);
    // /: Divisão (quociente) de dois números
    printf("x / y = %d\n", x / y);
    return 0;
}
```

```
x * y = 30
x / y = 3
```

Operadores Aritméticos - Binários

- Por que o resultado de `x / y` foi 3? Não deveria ser 3.333...?

Operadores Aritméticos - Binários

- Por que o resultado de `x / y` foi 3? Não deveria ser 3.333...?
- Hm... a formatação no comando de saída: `%d` vs `%f`

Operadores Aritméticos - Binários

- Por que o resultado de `x / y` foi 3? Não deveria ser 3.333...?
- Hm... a formatação no comando de saída: %d vs %f

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    int y = 3;
    // /: Divisão (quociente) de dois números
    printf("x / y = %f\n", x / y);
    return 0;
}
```

```
x / y = 0.000000
```

Operadores Aritméticos - Binários

- Por que o resultado de `x / y` foi 3? Não deveria ser 3.333...?
- Hm... a formatação no comando de saída: `%d` vs `%f`
- Agora o resultado foi 0???

Operadores Aritméticos - Binários

- Por que o resultado de `x / y` foi 3? Não deveria ser 3.333...?
- Hm... a formatação no comando de saída: `%d` vs `%f`
- Agora o resultado foi 0??? A operação foi realizada com dois **inteiros!**

Operadores Aritméticos - Binários

- Por que o resultado de `x / y` foi 3? Não deveria ser 3.333...?
- Hm... a formatação no comando de saída: `%d` vs `%f`
- Agora o resultado foi 0??? A operação foi realizada com dois **inteiros!**

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    int y = 3;
    // /: Divisão (quociente) de dois números
    printf("x / y = %f\n", (float) x / y);
    return 0;
}
```

```
x / y = 3.333333
```

Operadores Aritméticos - Binários

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    int y = 3;
    // /: Divisão (quociente) de dois números
    printf("x / y = %f\n", (float) x / y);
    // %: Módulo (resto da divisão) de dois números
    printf("x %% y = %d\n", x % y);
    return 0;
}
```

```
x / y = 3.333333
x % y = 1
```

Operadores Aritméticos - Unários

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    // +: mais unário ou positivo
    printf("+x = %d\n", +x);
    // -: menos unário ou negação
    printf("-x = %d\n", -x);
    return 0;
}
```

```
+x = 10
-x = -10
```

Operadores Aritméticos - Unários

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    // ++: pré ou pós incremento
    printf("++x = %d\n", ++x);
    printf("x++ = %d\n", x++);
    printf("x = %d\n", x);
    return 0;
}
```

```
++x = 11
x++ = 11
x = 12
```

Operadores Aritméticos - Unários

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    // ++: pré ou pós decremento
    printf("--x = %d\n", --x);
    printf("x-- = %d\n", x--);
    printf("x = %d\n", x);
    return 0;
}
```

```
--x = 9
x-- = 9
x = 8
```

Operadores Aritméticos - Unários

Diferença entre pré e pós incremento/decremento

- `y = x++`: incrementa depois de atribuir
- `y = ++x`: incrementa antes de atribuir

Expressões

- Expressões são combinações de variáveis, constantes, literais e operadores
- Exemplos:
  - `anos = dias / 365.25;`
  - `i = i + 3;`
  - `c = a * b + d / e;`
  - `c = a * (b + d) / e;`

Exercício 1

Escreva um programa que solicita ao usuário um inteiro de três algarismos e imprima na tela o seu valor invertido.

Exemplo

```
Entrada: 123
Valor invertido: 321
```

Exercício 1 - Solução

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int numero;
    scanf("%d", &numero);

    int unidade = numero % 10;
    int dezena = (numero / 10) % 10;
    int centena = numero / 100;

    int invertido = 100 * unidade + 10 * dezena + centena;
    printf("valor invertido: %d\n", invertido);
    return 0;
}
```

Conversão de Tipos (Cast)

- Força o resultado da expressão a ser de um tipo especificado
- `(tipo) expressão`
  - `(float) x`
  - `(int) x * 5.25`

Conversão de Tipos (Cast)

- Força o resultado da expressão a ser de um tipo especificado
- `(tipo) expressão`
  - `(float) x`
  - `(int) x * 5.25`

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    float f = x / 7;
    printf("%f\n", f);
    return 0;
}
```

```
1.0
```

Conversão de Tipos (Cast)

- Força o resultado da expressão a ser de um tipo especificado
- `(tipo) expressão`
  - `(float) x`
  - `(int) x * 5.25`

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    float f = (float) x / 7;
    printf("%f\n", f);
    return 0;
}
```

```
1.428571
```

Conversão de Tipos (Cast)

- Força o resultado da expressão a ser de um tipo especificado
- `(tipo) expressão`
  - `(float) x`
  - `(int) x * 5.25`
- O nível de prioridade (precedência) da conversão é maior que da divisão

Operadores Bit-a-Bit

- Operações bit-a-bit ajudam programadores que queiram trabalhar com o computador em "baixo nível"
- Essas operações só podem ser usadas nos tipos char, short, int e long
- O número é representado por sua forma binária e as operações são feitas em cada bit dele

Operadores Bit-a-Bit

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    char x = 10;
    char y = 3;
    // &: "E" ("And") bit-a-bit
    printf("x & y = %d\n", x & y);
    // |: "Ou" ("Or") bit-a-bit
    printf("x | y = %d\n", x | y);
    return 0;
}
```

```
x & y = 2
x | y = 11
```

Operadores Bit-a-Bit - & (E)

$$\begin{array}{rr} & 0000\ 1010 \\ \& & 0000\ 0011 \\ \hline & 0000\ 0010 \end{array}$$

- $0 \ \& \ 0 = 0$
- $0 \ \& \ 1 = 0$ (o mesmo para $1 \ \& \ 0$)
- $1 \ \& \ 1 = 1$

Operadores Bit-a-Bit - & (E)

$$\begin{array}{rr} & 0000\ 1010 \\ \& & 0000\ 0011 \\ \hline & 0000\ 0010 \end{array}$$

- $0 \,\&\, 0 = 0$
- $0 \,\&\, 1 = 0$ (o mesmo para $1 \,\&\, 0$)
- $1 \,\&\, 1 = 1$

Operadores Bit-a-Bit - & (E)

$$\begin{array}{rr} & 0000\ 1010 \\ \& & 0000\ 0011 \\ \hline & 0000\ 0010 \end{array}$$

- $0 \ \& \ 0 = 0$
- $0 \ \& \ 1 = 0$ (o mesmo para $1 \ \& \ 0$)
- $1 \ \& \ 1 = 1$

Operadores Bit-a-Bit - & (E)

$$\begin{array}{rr} & 0000\ 1010 \\ \& & 0000\ 0011 \\ \hline & 0000\ 0010 \end{array}$$

- $0 \,\&\, 0 = 0$
- **$0 \,\&\, 1 = 0$ (o mesmo para $1 \,\&\, 0$)**
- $1 \,\&\, 1 = 1$

Operadores Bit-a-Bit - | (Ou)

$$
\begin{array}{rr}
 & 0000\ 1010 \\
| & 0000\ 0011 \\
\hline
 & 0000\ 1011
\end{array}
$$

- $0 \mid 0 = 0$
- $0 \mid 1 = 1$ (o mesmo para $1 \mid 0$)
- $1 \mid 1 = 1$

Operadores Bit-a-Bit

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    char x = 10;
    char y = 3;
    // ^: "Ou exclusivo" ("Exclusive Or, xor") bit-a-bit
    printf("x ^ y = %d\n", x ^ y);
    // ~: Complemento bit-a-bit
    printf("~x = %d\n", ~x);
    return 0;
}
```

```
x ^ y = 9
~x = -11
```

Operadores Bit-a-Bit - $\wedge$ (Ou exclusivo)

$$
\begin{array}{rr}
 & 0000\ 1010 \\
\wedge & 0000\ 0011 \\
\hline
 & 0000\ 1001
\end{array}
$$

- $0 \wedge 0 = 0$
- $0 \wedge 1 = 1$ (o mesmo para $1 \wedge 0$)
- $1 \wedge 1 = 0$

Operadores Bit-a-Bit - $\sim$ (Complemento)

$$\begin{array}{ll} \sim & 0000\ 1010 \\ \hline & 1111\ 0101 \end{array}$$

- $\sim 0 = 1$
- $\sim 1 = 0$

Operadores Bit-a-Bit

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    char x = 10;
    char y = 3;
    // <<: Deslocamento a esquerda
    printf("x << y = %d\n", x << y);
    // >>: Deslocamento a direita
    printf("x >> y = %d\n", x >> y);
    return 0;
}
```

```
x << y = 80
x >> y = 1
```

Operadores Bit-a-Bit - Deslocamento

- $0000\ 1010 << 3 = 0101\ 0000$
- $0000\ 1010 >> 3 = 0000\ 0001$

Operadores Simplificados

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    // +=: soma e atribui, equivalente a x = x + 3
    x += 3;
    printf("x += 3: %d\n", x);
    // -=: subtrai e atribui, equivalente a x = x - 2
    x -= 2;
    printf("x -= 2: %d\n", x);
    return 0;
}
```

```
x += 3: 13
x -= 2: 11
```

Operadores Simplificados

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    // *=: multiplica e atribui, equivalente a x = x * 3
    x *= 3;
    printf("x *= 3: %d\n", x);
    // /=: divide e atribui quociente, equivalente a x = x / 2
    x /= 2;
    printf("x /= 2: %d\n", x);
    return 0;
}
```

```
x *= 3: 30
x /= 2: 15
```

Operadores Simplificados

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    // %=: divide e atribui resto, equivalente a x = x % 3
    x %= 3;
    printf("x %= 3: %d\n", x);
    // &=: "E" bit-a-bit e atribui, equivalente a x = x & 2
    x &= 2;
    printf("x &= 2: %d\n", x);
    return 0;
}
```

```
x %= 3: 1
x &= 2: 0
```

Operadores Simplificados

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    // |=: "Ou" bit-a-bit e atribui, equivalente a x = x | 3
    x |= 3;
    printf("x |= 3: %d\n", x);
    // <<=: shift a esquerda e atribui, equivalente a x = x << 2
    x <<= 2;
    printf("x <<= 2: %d\n", x);
    return 0;
}
```

```
x |= 3: 11
x <<= 2: 44
```

Overflow e Underflow

- Quando representamos um valor menor ou maior que o permitido pelo tipo, ocorre um erro de cálculo
  - Overflow: valor superior ao permitido
  - Underflow: valor inferior ao permitido

Underflow - Exemplo

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    short x = -32768;
    printf("Valor antes: %d\n", x);
    x--; // Gerando um underflow
    printf("Valor depois: %d\n", x);
    return 0;
}
```

```
Valor antes: -32768
Valor depois: 32767
```

Overflow - Exemplo

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    short x = 32767;
    printf("Valor antes: %d\n", x);
    x++; // Gerando um overflow
    printf("Valor depois: %d\n", x);
    return 0;
}
```

```
Valor antes: 32767
Valor depois: -32768
```

Overflow e Underflow

- Não dão erro: o programa **continua a execução** na **maioria** das linguagens de programação

Overflow e Underflow

- Não dão erro: o programa **continua a execução** na **maioria** das linguagens de programação
- **Moral da história:** procure saber quais são os valores máximos e mínimos que seu programa deve suportar, e escolha o tamanho da variável de acordo

Operadores Relacionais e Lógicos

- Operadores relacionais são utilizados na comparação de valores
- Esse tipo de operador retorna **verdadeiro** ou **falso**

Operadores Relacionais e Lógicos

- Operadores relacionais são utilizados na comparação de valores
- Esse tipo de operador retorna **verdadeiro** ou **falso**
  - Qual o tipo desses valores?

Operadores Relacionais e Lógicos

- Operadores relacionais são utilizados na comparação de valores
- Esse tipo de operador retorna **verdadeiro** ou **falso**
  - Qual o tipo desses valores? **lógico** ou **booleano**

Operadores Relacionais e Lógicos

- Operadores relacionais são utilizados na comparação de valores
- Esse tipo de operador retorna **verdadeiro** ou **falso**
  - Qual o tipo desses valores? **lógico** ou **booleano**
  - Mas C não possui um tipo **bool** por padrão, nem literais "verdadeiro" e "falso"
  - Logo, utilizamos o inteiro **1** para verdadeiro e **0** para falso

Operadores Relacionais e Lógicos

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    // <: menor
    printf("x < 10? %d\n", x < 10);
    // <=: menor ou igual
    printf("x <= 10? %d\n", x <= 10);
    return 0;
}
```

```
x < 10? 0
x <= 10? 1
```

Operadores Relacionais e Lógicos

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    // >: maior
    printf("x > 10? %d\n", x > 10);
    // >=: maior ou igual
    printf("x >= 10? %d\n", x >= 10);
    return 0;
}
```

```
x > 10? 0
x >= 10? 1
```

Operadores Relacionais e Lógicos

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int x = 10;
    // ==: igual
    printf("x == 10? %d\n", x == 10);
    // !=: diferente
    printf("x != 10? %d\n", x != 10);
    return 0;
}
```

```
x == 10? 1
x != 10? 0
```

Operadores Relacionais e Lógicos

- Existe uma biblioteca chamada **stdbool** que fornece o tipo **bool** e os valores **true** (verdadeiro) e **false** (falso)
- Podemos utilizá-la para tornar o código mais legível

Operadores Relacionais e Lógicos

```c
#include <stdio.h>
#include <stdbool.h>
int main(int argc, char *argv[]) {
    bool verdadeiro = true;
    printf("Verdadeiro: %d\n", verdadeiro);
    bool falso = false;
    printf("Falso: %d\n", falso);
    return 0;
}
```

```
Verdadeiro: 1
Falso: 0
```

Operadores Relacionais e Lógicos

- Operadores lógicos são utilizados para combinar valores lógicos (verdadeiro e falso)

Operadores Relacionais e Lógicos

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    char c = '5';
    // &&: "E" ("And") lógico
    printf("O caractere é um número? %d\n", c >= '0' && c <= '9');
    // ||: "Ou" ("Or") lógico
    printf("O caractere é a letra 'A'? %d\n", c == 'a' || c == 'A');
    return 0;
}
```

```
O caractere é um número? 1
O caractere é a letra 'A'? 0
```

Operadores Relacionais e Lógicos

```c
#include <stdio.h>
#include <stdbool.h>
int main(int argc, char *argv[]) {
    char c = '5';
    bool eh_numero = c >= '0' && c <= '9';
    // !: "Não" ("Not", negação)
    printf("O caractere *não* é um número? %d\n", !eh_numero);
    return 0;
}
```

```
O caractere *não* é um número? 0
```

Operadores Relacionais e Lógicos

a	b	!a	!b	a && b	a || b
F	F	V	V	F	F
F	V	V	F	F	V
V	F	F	V	F	V
V	V	F	F	V	V

Exercício 2

Escreva um programa que solicita ao usuário um inteiro e verifica se o número informado é par ou ímpar.

Exemplo

```
Entrada: 10
É ímpar? Não
```

Exercício 2 - Solução 1

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int numero;
    scanf("%d", &numero);
    printf("É ímpar? %d\n", numero % 2 != 0);
    return 0;
}
```

Exercício 2 - Solução 2

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    int numero;
    scanf("%d", &numero);
    printf("É ímpar? %d\n", numero & 1);
    return 0;
}
```