9002 — Aula 03
Algoritmos e Programação de Computadores

Instituto de Engenharia – UFMT

Segundo Semestre de 2014

29 de setembro de 2014

Agenda

Na aula passada...

Estudamos algumas questões relacionadas à resolução de problemas computacionais.

- O que é computador?
- Como interagir com o computador?
- O que é um algoritmo?
- ...

Na aula passada...

Estudamos algumas questões relacionadas à resolução de problemas computacionais.

- O que é computador?
- Como interagir com o computador?
- O que é um algoritmo?
- ...

Na aula passada...

Estudamos algumas questões relacionadas à resolução de problemas computacionais.

- O que é computador?
- Como interagir com o computador?
- O que é um algoritmo?
- ...

Na aula passada...

Estudamos algumas questões relacionadas à resolução de problemas computacionais.

- O que é computador?
- Como interagir com o computador?
- O que é um algoritmo?
- ...

Na aula passada...

Estudamos algumas questões relacionadas à resolução de problemas computacionais.

- O que é computador?
- Como interagir com o computador?
- O que é um algoritmo?
- ...

De algoritmos a programas

Visite http://light-bot.com/ :)

Mais especificamente...

Nos concentramos em alguns aspectos relacionados com 'conteúdo' e aprendemos:

- que um algoritmo, quando executado, associa **objetos de entrada** com **objetos de saída**, utilizando **objetos auxiliares** e de **controle**.
- a identificar estes objetos.
- a especificar nossos algoritmos utilizando estes objetos.

Hoje...

Estudaremos mais algumas questões.... :)

Formas de Representação

Existe um jeito único de representar um algoritmo? NÃO! Exemplos:

- Descrição narrativa
- Fluxograma
- Pseudo-código
- Linguagem de programação

Formas de Representação

Existe um jeito único de representar um algoritmo? NÃO! Exemplos:

- Descrição narrativa
- Fluxograma
- Pseudo-código
- Linguagem de programação

Formas de Representação

Existe um jeito único de representar um algoritmo? NÃO! Exemplos:

- Descrição narrativa
- Fluxograma
- Pseudo-código
- Linguagem de programação

De algoritmos a programas

- Como transformar um algoritmo em linguagem que o computador entenda?

Objetos

- Como os objetos de entrada, saída e auxiliares são definidos, especificados e manipulados na linguagem C?

Os objetos de entrada e saída, tal como vimos conceitualmente, são conhecidos nas linguagens de programação como 'variáveis'...

Variáveis

Definição

Variáveis são locais onde armazenamos valores. Toda variável é caracterizada por um nome, que a identifica em um programa e por um tipo, que determina o que pode ser armazenado naquela variável.

- Durante a execução do programa, um pedacinho da memória corresponde a variável.

Variáveis

Definição

Variáveis são locais onde armazenamos valores. Toda variável é caracterizada por um nome, que a identifica em um programa e por um tipo, que determina o que pode ser armazenado naquela variável.

- Durante a execução do programa, um pedacinho da memória corresponde a variável.

Declarando uma variável

Declara-se da seguinte forma: <**tipo_variável**> <**nome_variável**> **;**

Exemplos corretos

- int soma;
- float preco_abacaxi;
- char resposta;

Exemplos incorretos

- soma int;
- float preco_abacaxi

Declarando uma variável

Declara-se da seguinte forma: <**tipo_variável**> <**nome_variável**> **;**

Exemplos corretos

- int soma;
- float preco_abacaxi;
- char resposta;

Exemplos incorretos

- soma int;
- float preco_abacaxi

Declarando uma variável

Declara-se da seguinte forma: <**tipo_variável**> <**nome_variável**> **;**

Exemplos corretos

- int soma;
- float preco_abacaxi;
- char resposta;

Exemplos incorretos

- soma int;
- float preco_abacaxi

Variáveis inteiras

Variáveis inteiras são utilizadas para armazenar valores inteiros, em formato binário.

- Exemplo: $13_{10} = 1101_2$

Tipos inteiros em linguagem C

- **int:** Inteiro cujo comprimento depende do computador. É o inteiro mais utilizado. Em computadores *Pentium*, ocupa 32 bits e pode armazenar valores de -2.147.483.648 a 2.147.483.647.
- **unsigned int:** Inteiro cujo comprimento depende do computador e que armazena somente valores positivos. Em computadores *Pentium*, ocupa 32 bits e pode armazenar valores de 0 a 4.294.967.295.

Variáveis inteiras

Variáveis inteiras são utilizadas para armazenar valores inteiros, em formato binário.

- Exemplo: $13_{10} = 1101_2$

Tipos inteiros em linguagem C

- **int:** Inteiro cujo comprimento depende do computador. É o inteiro mais utilizado. Em computadores *Pentium*, ocupa 32 bits e pode armazenar valores de -2.147.483.648 a 2.147.483.647.
- **unsigned int:** Inteiro cujo comprimento depende do computador e que armazena somente valores positivos. Em computadores *Pentium*, ocupa 32 bits e pode armazenar valores de 0 a 4.294.967.295.

Variáveis inteiras

Mais tipos inteiros em linguagem C

- **long int:** Inteiro que ocupa 32 bits e pode armazenar valores de -2.147.483.648 a 2.147.483.647, independente do computador.
- **unsigned long int:** Inteiro que ocupa 32 bits e pode armazenar valores de 0 a 4.294.967.295, independente do computador.
- **short int:** Inteiro que ocupa 16 bits e pode armazenar valores de -32.768 a 32.767.
- **unsigned short int:** Inteiro que ocupa 16 bits e pode armazenar valores de 0 a 65.535.

Variáveis de tipo caractere

Variáveis utilizadas para armazenar letras e outros símbolos existentes em textos.

- Exemplos: `'a'`, `'A'`, `'8'`, `','`, `' '`, `'$'`, ...

Tipo de caractere em linguagem C

- **char:** Tipo caracter. Guarda apenas um caracter.

Variáveis de tipo caractere

Variáveis utilizadas para armazenar letras e outros símbolos existentes em textos.

- Exemplos: `'a'`, `'A'`, `'8'`, `','`, `' '`, `'$'`, ...

Tipo de caractere em linguagem C

- **char:** Tipo caracter. Guarda apenas um caracter.

Variáveis de tipo ponto flutuante

Armazenam valores reais, da seguinte forma de um produto e uma potência: $(-1)^{sinal} \cdot mantissa \cdot 2^{expoente}$.

- Exemplo: $0{,}5 = (-1)^0 \cdot 1 \cdot 2^{-1}$
- Para o programador, funciona como se ele armazenasse números na forma decimal.
- Possuem problemas de precisão (arredondamento).

Tipo de ponto (vírgula) flutuante em linguagem C

- **float:** Utiliza 32 bits, sendo 1 para o sinal, 8 para o expoente e 23 para a mantissa. Pode armazenar valores de $\pm 10^{-38}$ a $\pm 10^{38}$
- **double:** Utiliza 64 bits, sendo 1 para o sinal, 11 para o expoente e 52 para a mantissa. Pode armazenar valores de $\pm 10^{-308}$ a $\pm 10^{308}$

Variáveis de tipo ponto flutuante

Armazenam valores reais, da seguinte forma de um produto e uma potência: $(-1)^{sinal} \cdot mantissa \cdot 2^{expoente}$.

- Exemplo: $0{,}5 = (-1)^0 \cdot 1 \cdot 2^{-1}$
- Para o programador, funciona como se ele armazenasse números na forma decimal.
- Possuem problemas de precisão (arredondamento).

Tipo de ponto (vírgula) flutuante em linguagem C

- **float:** Utiliza 32 bits, sendo 1 para o sinal, 8 para o expoente e 23 para a mantissa. Pode armazenar valores de $\pm 10^{-38}$ a $\pm 10^{38}$
- **double:** Utiliza 64 bits, sendo 1 para o sinal, 11 para o expoente e 52 para a mantissa. Pode armazenar valores de $\pm 10^{-308}$ a $\pm 10^{308}$

Regras para nomes de variáveis em C

Regras para nomes de variáveis

- **Deve** começar com uma letra ou subcrito(_).
- **Nunca** pode começar com um número.
- Pode conter letras maíusculas, minúsculas, números e subscrito.
- Não se pode utilizar como parte do nome de uma variável:

```
{ ( + - * / \ ; . , ?
```

- Letras maiúsculas e minúsculas são diferentes:

```
int c;
int C;
```

Regras para nomes de variáveis em C

As seguintes palavras já tem um significado na linguagem C e por esse motivo não podem ser utilizadas como nome de variáveis:

```
auto      double    int       struct    break
enum      register  typedef   char      extern
return    union     const     float     short
unsigned  continue  for       signed    void
default   goto      sizeof    volatile  do
if        static    while
```

Constantes

Constantes são valores previamente determinados e que, por algum motivo, devem aparecer dentro de um programa.

- Assim como as variáveis, também possuem um tipo. Os tipos permitidos são exatamente os mesmos das variáveis, mais o tipo `string`, que corresponde a uma sequência de caracteres.
- Exemplos: `85, 0.10, 'c', "Hello, world!"`

Constantes

Exemplos de contantes

- **inteira:** é um número na forma decimal
  Exemplo: `10`, `145`, `1000000`
- **ponto flotuante:** é um número real, com parte fracionária
  Exemplo: `2.3456`, `32132131.5`, `5.0`
- **caractere:** é sempre representado por um símbolo entre aspas simples
  Exemplo: `'A'`
- **string:** é um texto entre aspas duplas
  Exemplo: `"Hello, world!"`

Comando de Atribuição

- O comando de atribuição em C é o sinal =
- A sintaxe do uso do comando é:

variável = valor ;

Exemplos

```
int a;
float c;
a = 5;
c = 67.89505456;
```

Comando de Atribuição

- O comando de atribuição em C é o sinal =
- A sintaxe do uso do comando é:

variável = valor ;

Exemplos

```
int a;
float c;
a = 5;
c = 67.89505456;
```

Comando de Atribuição

- O comando de atribuição pode conter expressões do lado direito:

variável = expressão ;

- Atribuir um valor de uma expressão a uma variável significa calcular o valor daquela expressão e copiar aquele valor para uma determinada variável.

Exemplos

```
int a;
float c;
a = 5+5+10;
c = 67.89505456+8-9;
```

Comando de Atribuição

- O comando de atribuição pode conter expressões do lado direito:

variável = expressão ;

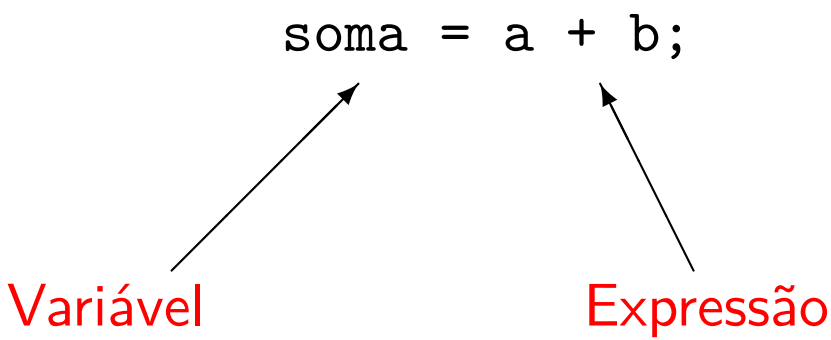
- Atribuir um valor de uma expressão a uma variável significa calcular o valor daquela expressão e copiar aquele valor para uma determinada variável.

Exemplos

```
int a;
float c;
a = 5+5+10;
c = 67.89505456+8-9;
```

Comando de Atribuição

No exemplo abaixo, a variável **soma** recebe o valor calculado da expressão **a + b**



Atribuição

- O operador de atribuição é o sinal de igual (=)

À esquerda do operador de atribuição deve existir somente o nome de uma **variável**.

=

À direita, deve haver uma **expressão** cujo valor será calculado e armazenado na variável

Expressões Simples

Expressões simples

- Uma constante é uma expressão:
  Exemplo: `a = 10;`
- Uma variável é uma expressão:
  Exemplo: `a = b;`

Exemplos de atribuição

Sempre que uma variável for usada, ela deve ter sido **declarada antes**.

Exemplos

```
int a,b;
float f,g;
char h;

a = 10;
b = -15;
f = 10.0;
h = 'A';

a = b;
f = a;
a = (b+f+a);
```

Exemplos de atribuição

Sempre que uma variável for usada, ela deve ter sido **declarada antes**.

Exemplos

```
int a,b;
float f,g;
char h;

a = 10;
b = -15;
f = 10.0;
h = 'A';

a = b;
f = a;
a = (b+f+a);
```

Exemplos errados de atribuição

Exemplos errados

```
int a, b;
float f, g;
char h;

a b = 10;
b = -15
d = 90;
```

Estrutura Básica de um Programa em C

Estrutura básica de um programa C

Declaração de bibliotecas Usadas

Declaração de variáveis

```
int main(){
   Declaração de variáveis

   Comandos

   ...

   Comandos
   return 0;
}
```

Estrutura Básica de um Programa em C

Exemplo

```
#include <stdio.h>

int main(){
    int a;
    int b, c;

    a = 7 + 9;
    b = a + 10;
    c = b - a;

    return 0;
}
```

Informações Extras: Constantes Inteiras

- Um número na forma decimal é escrito normalmente
  Ex: `10`, `145`, `1000000`
- Um número na forma hexadecimal (base 16) é precedido de 0x
  Ex: `0xA` ($A_{16} = 10$), `0x100` ($100_{16} = 256$)
- Um número na forma octal (base 8) é precedido de 0
  Ex: `010` ($10_8 = 8$)

Informações Extras: Constantes do tipo de ponto flutuante

- Em C, um número só é considerado ponto flutuante ou fracionário se tiver uma parte "não inteira", mesmo que essa parte tenha valor zero. Utilizamos o ponto para separação:
  Ex: `10.0`, `5.2`, `3569.22565845`
- Um número inteiro ou decimal seguido da letra e e um número é interpretado como:

$$numero \cdot 10^{expoente}$$

Ex: $2e2 = 2 \cdot 10^2 = 200.0$

Informações Extras: Caracter

- Caracteres são, na verdade, variáveis inteiras que armazenam um número associado ao símbolo. A convenção normalmente adota é a tabela ASCII (*American Standard Code for Information Interchang*), mas existem outras (EBCDIC, Unicode etc.).
- `char`: Armazena um símbolo (no caso, o inteiro correspondente). Seu valor pode ir de -128 a 127.
- `unsigned char`: Armazena um símbolo (no caso, o inteiro correspondente). Seu valor pode ir de 0 a 255.
- Toda constante do tipo caracter pode ser usada como uma constante do tipo inteiro. Nesse caso, o valor atribuído será o valor daquela letra na tabela ASCII.

Informações Extras: Tabela ASCII

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0 16	Caracteres de Controle															
32		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	/	]	⋀	_
96	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Informações Extras: Obtendo o tamanho de um tipo

O comando `sizeof(tipo)` retorna o tamanho, em bytes, de um determinado tipo. (Um byte corresponde a 8 bits).

Exemplo

```
a = sizeof(int);
```

Variável a recebe o valor 4 (32 bits).

Exercício

Complete e corrija o código abaixo.

Tipos

```
int main() {
    10 = a;
    b = -6;
    c = 100000;
    d = 33000.;
    e = -80000,657;
    f = 30;
    g = a;
    h = 'a';
}
```

Expressões

- Já vimos que constantes e variáveis são expressões.
- Uma expressão também pode ser é um conjunto de operações aritméticas, lógicas ou relacionais utilizados para fazer "cálculos" sobre os valores das variáveis.

Exemplo

```
a + b
```

Calcula a soma de a e b

Expressões Aritméticas

Os operadores aritméticos são: +,-,*,/

Expressões

- <expressao> + <expressao>: Soma
  Ex: `a = a + b;`
- <expressao> - <expressao>: Diferença
  Ex: `a = a - b;`
- <expressao> * <expressao>: Produto
  Ex: `a = a * b;`
- <expressao> / <expressao>: Divisão
  Ex: `a = a / b;`
- <expressao> % <expressao>: Resto da divisão inteira
  Ex: `a = a % b;`
- - <expressao>: Inverte o sinal
  Ex: `a = -b;`

Expressões Aritméticas

Os operadores aritméticos são: +,-,*,/

Expressões

- <expressao> + <expressao>: Soma
  Ex: `a = a + b;`
- <expressao> - <expressao>: Diferença
  Ex: `a = a - b;`
- <expressao> * <expressao>: Produto
  Ex: `a = a * b;`
- <expressao> / <expressao>: Divisão
  Ex: `a = a / b;`
- <expressao> % <expressao>: Resto da divisão inteira
  Ex: `a = a % b;`
- - <expressao>: Inverte o sinal
  Ex: `a = -b;`

Expressões Aritméticas

Os operadores aritméticos são: +,-,*,/

Expressões

- <expressao> + <expressao>: Soma
  Ex: `a = a + b;`
- <expressao> - <expressao>: Diferença
  Ex: `a = a - b;`
- <expressao> * <expressao>: Produto
  Ex: `a = a * b;`
- <expressao> / <expressao>: Divisão
  Ex: `a = a / b;`
- <expressao> % <expressao>: Resto da divisão inteira
  Ex: `a = a % b;`
- - <expressao>: Inverte o sinal
  Ex: `a = -b;`

Expressões Aritméticas

Os operadores aritméticos são: +,-,*,/

Expressões

- <expressao> + <expressao>: Soma
  Ex: `a = a + b;`
- <expressao> - <expressao>: Diferença
  Ex: `a = a - b;`
- <expressao> * <expressao>: Produto
  Ex: `a = a * b;`
- <expressao> / <expressao>: Divisão
  Ex: `a = a / b;`
- <expressao> % <expressao>: Resto da divisão inteira
  Ex: `a = a % b;`
- - <expressao>: Inverte o sinal
  Ex: `a = -b;`

Expressões Aritméticas

Os operadores aritméticos são: +,-,*,/

Expressões

- <expressao> + <expressao>: Soma
  Ex: `a = a + b;`
- <expressao> - <expressao>: Diferença
  Ex: `a = a - b;`
- <expressao> * <expressao>: Produto
  Ex: `a = a * b;`
- <expressao> / <expressao>: Divisão
  Ex: `a = a / b;`
- <expressao> % <expressao>: Resto da divisão inteira
  Ex: `a = a % b;`
- - <expressao>: Inverte o sinal
  Ex: `a = -b;`

Expressões Aritméticas

Os operadores aritméticos são: +,-,*,/

Expressões

- <expressao> + <expressao>: Soma
  Ex: `a = a + b;`
- <expressao> - <expressao>: Diferença
  Ex: `a = a - b;`
- <expressao> * <expressao>: Produto
  Ex: `a = a * b;`
- <expressao> / <expressao>: Divisão
  Ex: `a = a / b;`
- <expressao> % <expressao>: Resto da divisão inteira
  Ex: `a = a % b;`
- - <expressao>: Inverte o sinal
  Ex: `a = -b;`

Expressões Aritméticas

Os operadores aritméticos são: +,-,*,/

Expressões

- <expressao> + <expressao>: Soma
  Ex: `a = a + b;`
- <expressao> - <expressao>: Diferença
  Ex: `a = a - b;`
- <expressao> * <expressao>: Produto
  Ex: `a = a * b;`
- <expressao> / <expressao>: Divisão
  Ex: `a = a / b;`
- <expressao> % <expressao>: Resto da divisão inteira
  Ex: `a = a % b;`
- - <expressao>: Inverte o sinal
  Ex: `a = -b;`

Expressões

Criando expressões

- Operadores aritméticas (e *todos* os demais) juntam várias expressões.
- Podemos formar expressões complexas combinando vários operadores
  Exemplo: `a = -b + 2 + c - (9 + d * 8)`

Ordem de avaliação

- Qual o valor da expressão `5 + 10 % 3`?
- E da expressão `5 * 10 % 3`?

Expressões

Criando expressões

- Operadores aritméticas (e *todos* os demais) juntam várias expressões.
- Podemos formar expressões complexas combinando vários operadores
  Exemplo: `a = -b + 2 + c - (9 + d * 8)`

Ordem de avaliação

- Qual o valor da expressão `5 + 10 % 3`?
- E da expressão `5 * 10 % 3`?

Expressões

Criando expressões

- Operadores aritméticas (e *todos* os demais) juntam várias expressões.
- Podemos formar expressões complexas combinando vários operadores
  Exemplo: `a = -b + 2 + c - (9 + d * 8)`

Ordem de avaliação

- Qual o valor da expressão `5 + 10 % 3`?
- E da expressão `5 * 10 % 3`?

Expressões

Criando expressões

- Operadores aritméticas (e *todos* os demais) juntam várias expressões.
- Podemos formar expressões complexas combinando vários operadores
  Exemplo: `a = -b + 2 + c - (9 + d * 8)`

Ordem de avaliação

- Qual o valor da expressão `5 + 10 % 3`?
- E da expressão `5 * 10 % 3`?

Expressões

Criando expressões

- Operadores aritméticas (e *todos* os demais) juntam várias expressões.
- Podemos formar expressões complexas combinando vários operadores
  Exemplo: `a = -b + 2 + c - (9 + d * 8)`

Ordem de avaliação

- Qual o valor da expressão `5 + 10 % 3`?
- E da expressão `5 * 10 % 3`?

Expressões

Criando expressões

- Operadores aritméticas (e *todos* os demais) juntam várias expressões.
- Podemos formar expressões complexas combinando vários operadores
  Exemplo: `a = -b + 2 + c - (9 + d * 8)`

Ordem de avaliação

- Qual o valor da expressão `5 + 10 % 3`?
- E da expressão `5 * 10 % 3`?

Precedência

Precedência

Precedência define a ordem em que os operadores serão calculados quando o programa for executado.

Regras de precedência

Em C, os operadores são calculados na seguinte ordem:

1. *, % e /, na ordem em que aparecerem na expressão
2. + e -, na ordem em que aparecerem na expressão

Exemplo: 8 + 10 * 6 vale 68.

Precedência

Precedência

Precedência define a ordem em que os operadores serão calculados quando o programa for executado.

Regras de precedência

Em C, os operadores são calculados na seguinte ordem:

1. *, % e /, na ordem em que aparecerem na expressão
2. + e -, na ordem em que aparecerem na expressão

Exemplo: 8 + 10 * 6 vale 68.

Precedência

Precedência

Precedência define a ordem em que os operadores serão calculados quando o programa for executado.

Regras de precedência

Em C, os operadores são calculados na seguinte ordem:

1. *, % e /, na ordem em que aparecerem na expressão
2. + e -, na ordem em que aparecerem na expressão

Exemplo: 8 + 10 * 6 vale 68.

Precedência

Precedência

Precedência define a ordem em que os operadores serão calculados quando o programa for executado.

Regras de precedência

Em C, os operadores são calculados na seguinte ordem:

1. *, **%** e /, na ordem em que aparecerem na expressão
2. + e -, na ordem em que aparecerem na expressão

Exemplo: 8 + 10 * 6 vale 68.

Precedência

Precedência

Precedência define a ordem em que os operadores serão calculados quando o programa for executado.

Regras de precedência

Em C, os operadores são calculados na seguinte ordem:

1. `*`, `%` e `/`, na ordem em que aparecerem na expressão
2. `+` e `-`, na ordem em que aparecerem na expressão

Exemplo: `8 + 10 * 6` vale 68.

Alterando a precedência

Parênteses

Para alterara precedência devemos usar parênteses:

- (<**expressao**>) é uma expressão com o valor da expressão interna
  Exemplo: 5 + 10 % 3 retorna 6, mas (5 + 10) % 3 retorna 0
- **Regra da paridade:**
  - número de parênteses (que abrem deve ser **igual** ao
  - número de parênteses) que fecham expressões

Dica: Devemos usar parênteses sempre que não é claro qual é a ordem de avaliação de uma expressão.

Alterando a precedência

Parênteses

Para alterara precedência devemos usar parênteses:

- **(**<**expressao**>**)** é uma expressão com o valor da expressão interna
  Exemplo: 5 + 10 % 3 retorna 6, mas (5 + 10) % 3 retorna 0
- **Regra da paridade:**
  - número de parênteses (que abrem deve ser **igual** ao
  - número de parênteses) que fecham expressões

Dica: Devemos usar parênteses sempre que não é claro qual é a ordem de avaliação de uma expressão.

Alterando a precedência

Parênteses

Para alterara precedência devemos usar parênteses:

- **(**<**expressao**>**)** é uma expressão com o valor da expressão interna
  Exemplo: `5 + 10 % 3` retorna 6, mas `(5 + 10) % 3` retorna 0
- **Regra da paridade:**
  - número de parênteses **(** que abrem deve ser **igual** ao
  - número de parênteses **)** que fecham expressões

Dica: Devemos usar parênteses sempre que não é claro qual é a ordem de avaliação de uma expressão.

Alterando a precedência

Parênteses

Para alterara precedência devemos usar parênteses:

- **(**<**expressao**>**)** é uma expressão com o valor da expressão interna
  Exemplo: `5 + 10 % 3` retorna 6, mas `(5 + 10) % 3` retorna 0
- **Regra da paridade:**
  - número de parênteses (que abrem deve ser **igual** ao
  - número de parênteses) que fecham expressões

Dica: Devemos usar parênteses sempre que não é claro qual é a ordem de avaliação de uma expressão.

Alterando a precedência

Parênteses

Para alterara precedência devemos usar parênteses:

- **(**<**expressao**>**)** é uma expressão com o valor da expressão interna
  Exemplo: `5 + 10 % 3` retorna 6, mas `(5 + 10) % 3` retorna 0
- **Regra da paridade:**
  - número de parênteses (que abrem deve ser **igual** ao
  - número de parênteses) que fecham expressões

Dica: Devemos usar parênteses sempre que não é claro qual é a ordem de avaliação de uma expressão.

Incremento(++) e Decremento(−−)

Operadores de incremento

Algumas operação bastante comuns têm atalhos em C:

- Incremento de variável: somam uma unidade a uma variável
  Exemplo: $c{+}{+}$
- Decremento de variável: diminuem uma unidade de uma variável
  Exemplo: $c - -$

Atenção: Dependendo da posição do operador de incremento, a expresssão retorna um valor **antes** ou **depois** de alterar a variável.

Incremento(++) e Decremento(−−)

Operadores de incremento

Algumas operação bastante comuns têm atalhos em C:

- Incremento de variável: somam uma unidade a uma variável
  Exemplo: **c++**
- Decremento de variável: diminuem uma unidade de uma variável
  Exemplo: $c--$

Atenção: Dependendo da posição do operador de incremento, a expresssão retorna um valor **antes** ou **depois** de alterar a variável.

Incremento(++) e Decremento(−−)

Operadores de incremento

Algumas operação bastante comuns têm atalhos em C:

- Incremento de variável: somam uma unidade a uma variável
  Exemplo: $c{+}{+}$
- Decremento de variável: diminuem uma unidade de uma variável
  Exemplo: $c--$

Atenção: Dependendo da posição do operador de incremento, a expresssão retorna um valor **antes** ou **depois** de alterar a variável.

Incremento(++) e Decremento(−−)

Operadores de incremento

Algumas operação bastante comuns têm atalhos em C:

- Incremento de variável: somam uma unidade a uma variável
  Exemplo: **c++**
- Decremento de variável: diminuem uma unidade de uma variável
  Exemplo: $c--$

Atenção: Dependendo da posição do operador de incremento, a expresssão retorna um valor **antes** ou **depois** de alterar a variável.

Incremento(++) e Decremento(−−)

Para incrementar uma variável **antes** de retornar o seu valor colocamos o operador à esquerda:

Operador à esquerda da variável:

```
#include <stdio.h>

int main (void) {
    int a = 10;
    printf ("%d", ++a);
}
```

Imprime 11

Incremento(++) e Decremento(−−)

Para incrementar uma variável **antes** de retornar o seu valor colocamos o operador à esquerda:

Operador à esquerda da variável:

```
#include <stdio.h>

int main (void) {
    int a = 10;
    printf ("%d", ++a);
}
```

Imprime 11

Incremento(++) e Decremento(−−)

Para incrementar uma variável **antes** de retornar o seu valor colocamos o operador à esquerda:

Operador à esquerda da variável:

```
#include <stdio.h>

int main (void) {
    int a = 10;
    printf ("%d", ++a);
}
```

Imprime 11

Incremento(++) e Decremento(−−)

Para primeiro retornar o valor da variável e **depois** incrementar uma variável colocamos o operador à direita:

Operador à direita da variável:

```
#include <stdio.h>

int main (void) {
    int a = 10;
    printf ("%d", a++);
}
```

Imprime 10

Incremento(++) e Decremento(−−)

Para primeiro retornar o valor da variável e **depois** incrementar uma variável colocamos o operador à direita:

Operador à direita da variável:

```
#include <stdio.h>

int main (void) {
    int a = 10;
    printf ("%d", a++);
}
```

Imprime 10

Incremento(++) e Decremento(−−)

Para primeiro retornar o valor da variável e **depois** incrementar uma variável colocamos o operador à direita:

Operador à direita da variável:

```
#include <stdio.h>

int main (void) {
    int a = 10;
    printf ("%d", a++);
}
```

Imprime 10

Incremento(++) e Decremento(−−)

Em uma expressão, os operadores de incremento e decremento são sempre calculados **antes** dos demais operadores (têm maior precedência)

Expressões confusas:

```
#include <stdio.h>

int main (void) {
    int a = 10;
    printf ("%d", a * ++a);
}
```

Imprime 121

Atenção: Devemos **evitar** utilizar expressões confusas como essas.

Incremento(++) e Decremento(−−)

Em uma expressão, os operadores de incremento e decremento são sempre calculados **antes** dos demais operadores (têm maior precedência)

Expressões confusas:

```
#include <stdio.h>

int main (void) {
    int a = 10;
    printf ("%d", a * ++a);
}
```

Imprime 121

Atenção: Devemos **evitar** utilizar expressões confusas como essas.

Incremento(++) e Decremento(−−)

Em uma expressão, os operadores de incremento e decremento são sempre calculados **antes** dos demais operadores (têm maior precedência)

Expressões confusas:

```
#include <stdio.h>

int main (void) {
    int a = 10;
    printf ("%d", a * ++a);
}
```

Imprime 121

Atenção: Devemos **evitar** utilizar expressões confusas como essas.

Incremento(++) e Decremento(−−)

Em uma expressão, os operadores de incremento e decremento são sempre calculados **antes** dos demais operadores (têm maior precedência)

Expressões confusas:

```
#include <stdio.h>

int main (void) {
    int a = 10;
    printf ("%d", a * ++a);
}
```

Imprime 121

Atenção: Devemos **evitar** utilizar expressões confusas como essas.

Atribuições simplificadas

Atalhos

Podemos simplificar algumas expressões comuns em C:

- Uma expressão da forma

```
a = a + b;
```

- podem ser escritas de maneira **equivalente** como

```
a += b;
```

Atribuições simplificadas

Atalhos

Podemos simplificar algumas expressões comuns em C:

- Uma expressão da forma

```
a = a + b;
```

- podem ser escritas de maneira **equivalente** como

```
a += b;
```

Atribuições simplificadas

Atalhos

Podemos simplificar algumas expressões comuns em C:

- Uma expressão da forma

  ```
  a = a + b;
  ```

- podem ser escritas de maneira **equivalente** como

  ```
  a += b;
  ```

Atribuições simplificadas

Tabela de atribuições simplificadas

Comando	Exemplo	Correspondente a
+=	`a += b;`	`a = a + b;`
-=	`a -= b;`	`a = a - b;`
*=	`a *= b;`	`a = a * b;`
/=	`a /= b;`	`a = a / b;`
%=	`a %= b;`	`a = a % b;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação
  Exemplo 1: `int a; short b; a = b;`
  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação
  Exemplo correto: `a = (int)( (float)b / (float)c );`
  Exemplo **incorreto**: `int a; (float)a=1.0;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação
  Exemplo 1: `int a; short b; a = b;`
  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação
  Exemplo correto: `a = (int)( (float)b / (float)c );`
  Exemplo **incorreto**: `int a; (float)a=1.0;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação
  Exemplo 1: `int a; short b; a = b;`
  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação
  Exemplo correto: `a = (int)( (float)b / (float)c );`
  Exemplo **incorreto**: `int a; (float)a=1.0;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação

  Exemplo 1: `int a; short b; a = b;`

  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação

  Exemplo correto: `a = (int)( (float)b / (float)c );`

  Exemplo **incorreto**: `int a; (float)a=1.0;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação

  Exemplo 1: `int a; short b; a = b;`

  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação

  Exemplo correto: `a = (int)( (float)b / (float)c );`

  Exemplo **incorreto**: `int a; (float)a=1.0;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação

  Exemplo 1: `int a; short b; a = b;`

  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação

  Exemplo correto: `a = (int)( (float)b / (float)c );`

  Exemplo **incorreto**: `int a; (float)a=1.0;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação
  Exemplo 1: `int a; short b; a = b;`
  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação
  Exemplo correto: `a = (int)( (float)b / (float)c );`
  Exemplo **incorreto**: `int a; (float)a=1.0;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação

  Exemplo 1: `int a; short b; a = b;`

  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação

  Exemplo correto: `a = (int)( (float)b / (float)c );`

  Exemplo **incorreto**: `int a; (float)a=1.0;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação
  Exemplo 1: `int a; short b; a = b;`
  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação
  Exemplo correto: `a = (int)( (float)b / (float)c );`
  Exemplo **incorreto**: `int a; (float)a=1.0;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação

  Exemplo 1: `int a; short b; a = b;`

  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação

  Exemplo correto: `a = (int)( (float)b / (float)c );`

  Exemplo **incorreto**: `int a; (float)a=1.0;`

Conversão de tipos

Convertendo tipos

- Para obter um valor em um tipo diferente podemos alguns tipos
- Existem duas maneiras: **implícita** e **explícita**.

Conversão implícita

- O tamanho de destino é maior do que de origem
- Não há perda de informação
  Exemplo 1: `int a; short b; a = b;`
  Exemplo 2: `float a; int b=10; a = b;`

Conversão explícita

- O tipo de destino é informado **explicitamente**
- Pode haver perda de informação
  Exemplo correto: `a = (int)( (float)b / (float)c );`
  Exemplo **incorreto**: `int a; (float)a=1.0;`

Um uso da conversão de tipos

Operação de divisão

O operador de divisão / possui dois modos:

1. **Inteiro:** se os dois argumentos forem inteiros
   Exemplo: a expressão `10 / 3` vale `3`.
2. **Ponto flutuante:** se pelo menos um argumento for flutuante
   Exemplo: a expressão `1.5 / 3` vale `0.5`.

Exemplo de conversão de tipos

```
int main() {
  int a, b;
  float f;
  a = 10;
  b = 3;
  f = a / (float) b;
}
```

O valor de `f` será **3.33333333**.

Um uso da conversão de tipos

Operação de divisão

O operador de divisão / possui dois modos:

1. **Inteiro:** se os dois argumentos forem inteiros
   Exemplo: a expressão 10 / 3 vale 3.
2. **Ponto flutuante:** se pelo menos um argumento for flutuante
   Exemplo: a expressão 1.5 / 3 vale 0.5.

Exemplo de conversão de tipos

```
int main() {
  int a, b;
  float f;
  a = 10;
  b = 3;
  f = a / (float) b;
}
```

O valor de f será **3.33333333**.

Um uso da conversão de tipos

Operação de divisão

O operador de divisão / possui dois modos:

1. **Inteiro:** se os dois argumentos forem inteiros
   Exemplo: a expressão `10 / 3` vale 3.
2. **Ponto flutuante:** se pelo menos um argumento for flutuante
   Exemplo: a expressão `1.5 / 3` vale `0.5`.

Exemplo de conversão de tipos

```
int main() {
  int a, b;
  float f;
  a = 10;
  b = 3;
  f = a / (float) b;
}
```

O valor de `f` será **3.33333333**.

Um uso da conversão de tipos

Operação de divisão

O operador de divisão / possui dois modos:

1. **Inteiro:** se os dois argumentos forem inteiros
   Exemplo: a expressão `10 / 3` vale 3.
2. **Ponto flutuante:** se pelo menos um argumento for flutuante
   Exemplo: a expressão `1.5 / 3` vale `0.5`.

Exemplo de conversão de tipos

```
int main() {
  int a, b;
  float f;
  a = 10;
  b = 3;
  f = a / (float) b;
}
```

O valor de f será 3.33333333.

Um uso da conversão de tipos

Operação de divisão

O operador de divisão / possui dois modos:

1. **Inteiro:** se os dois argumentos forem inteiros
   Exemplo: a expressão `10 / 3` vale 3.
2. **Ponto flutuante:** se pelo menos um argumento for flutuante
   Exemplo: a expressão `1.5 / 3` vale 0.5.

Exemplo de conversão de tipos

```
int main() {
  int a, b;
  float f;
  a = 10;
  b = 3;
  f = a / (float) b;
}
```

O valor de `f` será **3.33333333**.

Exercícios

```
#include <stdio.h>
main(void){
    int n, k;
    n = 5;
    k = 1 + 2 * 30 / (++n % 4 * -5);
}
```

1. Indique a ordem em que são executadas as operações.
2. Reescreva o programa de forma que cada comando contenha apenas uma operação aritmética.

Nas próximas aulas...

- Nas próximas aulas veremos como interagir com os dispositivos de entrada e saída do computador.
- FIM!!!