

# MC102 – Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Segundo Semestre de 2013

# Roteiro

- 1 Indentação
- 2 Comentários
- 3 Saída de dados
- 4 Entrada de dados
- 5 Expressões aritméticas
- 6 Conversão de tipos

# Indentação

- A indentação refere-se ao espaçamento ou tabulação inserida no início das linhas no código fonte do programa.
- Seu objetivo é indicar que os elementos (blocos de comandos) hierarquicamente dispostos possuem o mesmo espaçamento.
- Embora modifique o código apenas do ponto de vista estético, a indentação facilita a leitura e interpretação do programa.

## Exemplo de programa não indentado:

```
#include <stdio.h>
int main() { printf("Hello, world!\n"); return 0; }
```

## Exemplo de programa indentado:

```
#include <stdio.h>
int main() {
    printf("Hello, world!\n");
    return 0;
}
```

## Comentários

- Um programa pode conter comentários, que servem para auxiliar quem for ler o programa, mas que são ignorados pelo compilador.
- Há duas formas de inserir um comentário em um programa C:
  - ▶ Comentários delimitados pelos símbolos /\* e \*/ podem se estender por múltiplas linhas. Forma padrão de comentários em C.
  - ▶ O símbolo // pode ser usado para inserir um comentário mais curto, de uma única linha. Forma não-padrão: deve ser evitada.

### Exemplo:

```
#include <stdio.h>

/* Meu primeiro programa:
   Este programa imprime uma mensagem na saída padrão. */
int main() {
    printf("Hello, world!\n"); // Imprime a mensagem
    return 0;
}
```

## Imprimindo uma mensagem

- Pode-se imprimir um texto utilizando o comando `printf`. O texto pode ser uma constante do tipo `string`.

### Exemplo:

```
printf("Ola Pessoal!");  
printf("Tudo bem?");
```

Saída:

Ola Pessoal!Tudo bem?

- No meio da constante `string` pode haver comandos especiais. O símbolo especial `\n` é responsável por pular uma linha na saída.

### Exemplo:

```
printf("Ola Pessoal!\nTudo bem?\n");
```

Saída:

Ola Pessoal!  
Tudo bem?

## Imprimindo o conteúdo de uma variável

- Pode-se imprimir, além de texto simples, o conteúdo de uma variável utilizando o comando `printf`. Para isso, utiliza-se símbolos especiais no texto para indicar que aquele trecho deve ser substituído por uma variável e, no final, passa-se uma lista de variáveis ou constantes, separadas por vírgula.

### Exemplo:

```
int x = 10;  
printf("A variavel %c contem o valor %d.\n", 'x', x);
```

Saída:

A variavel x contem o valor 10.

- Nesse caso, `%c` deve ser substituído por uma variável ou constante do tipo `char`, enquanto `%d` deve ser substituído por uma variável ou constante do tipo `int`.

# Formatos inteiros

`%d` — Imprime um valor inteiro.

## Exemplo:

```
printf("%d anos\n", 10);
```

Saída:

10 anos

## Exemplo:

```
int a = 12;  
printf("Valor registrado: %d\n", a);
```

Saída:

Valor registrado: 12

## Formatos inteiros

- O argumento `d` pode ser substituído pelos argumentos `u`, `ld` e `lu`, quando se deseja imprimir variáveis do tipo `unsigned int`, `long int` ou `unsigned long int`, respectivamente.

### Exemplo:

```
printf("%d\n", 4000000000);
```

Saída:

-294967296

### Exemplo:

```
printf("%ld\n", 4000000000);
```

Saída:

4000000000

## Formatos inteiros

- O argumento `d` pode ser substituído pelos argumentos `u`, `ld` e `lu`, quando se deseja imprimir variáveis do tipo `unsigned int`, `long int` ou `unsigned long int`, respectivamente.

### Exemplo:

```
printf("%u\n", 3000000000 + 3000000000);
```

Saída:

1705032704

### Exemplo:

```
printf("%lu\n", 3000000000 + 3000000000);
```

Saída:

6000000000

# Formatos ponto flutuante

`%f` — Imprime um valor em ponto flutuante.

Exemplo:

```
printf("%f\n", 10.0);
```

Saída:

```
10.000000
```

# Formatos ponto flutuante

`%.<decimais>f` — Imprime valor em um ponto flutuante, com `<decimais>` casas decimais.

## Exemplo:

```
printf("Valor Total: R$%.2f\n", 195.739);
```

Saída:

```
Valor Total: R$195.74
```

## Formatos ponto flutuante

Também é possível especificar pelo menos quantos caracteres serão impressos antes do ponto (caso o número não possua dígitos suficientes são usados espaços em branco para completar a impressão).

### Exemplo:

```
printf("%6.2f\n", 10.0);
```

Saída:

```
10.00
```

# Formatos ponto flutuante

`%e` — Imprime um valor em ponto flutuante, em notação científica.

## Exemplo:

```
printf("%e\n", 10.02545);
```

Saída:

```
1.002545e+01
```

## Formato caractere

`%c` — Imprime um caractere.

### Exemplo:

```
printf("%c\n", 'A');
```

Saída:

A

### Exemplo:

```
printf("%d\n", 'A');
```

Saída:

65

### Exemplo:

```
printf("%c\n", 'b' + 3);
```

Saída:

e

# Formato string

`%s` — Imprime uma string (cadeia de caracteres).

## Exemplo:

```
printf("Meu %s programa\n", "primeiro");
```

Saída:

```
Meu primeiro programa
```

# A função scanf

- Realiza a leitura de um texto a partir do teclado.
- Parâmetros:
  - ▶ Uma string, indicando os tipos das variáveis que serão lidas e o formato dessa leitura (Ex.: “%d”, “%f”, “%c”, etc).
  - ▶ Uma lista de variáveis (Ex.: &idade, &valor, &letra, etc).
- Aguarda que o usuário digite um valor e atribui o valor digitado à variável.

## A função scanf

O programa abaixo é composto de quatro passos:

- 1 Cria uma variável `n`
- 2 Imprime a mensagem "Digite um numero:"
- 3 Lê o valor do número digitado
- 4 Imprime o valor do número digitado

```
#include <stdio.h>
```

```
int main() {  
    int n;  
    printf("Digite um numero: ");  
    scanf("%d", &n); /* Note o simbolo & */  
    printf("O valor digitado foi %d\n", n);  
    return 0;  
}
```

## A função scanf

Leitura de múltiplas variáveis:

```
#include <stdio.h>
```

```
int main() {  
    int m, n, o;  
    printf("Digite tres numeros: ");  
    scanf("%d %d %d", &m, &n, &o); /* Note o simbolo & */  
    printf("Os valores digitados foram %d %d %d\n", m, n, o);  
    return 0;  
}
```

## A função scanf

Leitura de múltiplas variáveis separadas por vírgulas:

```
#include <stdio.h>
```

```
int main() {  
    int m, n, o;  
    printf("Digite tres numeros (separados por virgulas): ");  
    scanf("%d,%d,%d", &m, &n, &o); /* Note o simbolo & */  
    printf("Os valores digitados foram %d %d %d\n", m, n, o);  
    return 0;  
}
```

## Formatos de leitura de variável

Os formatos de leitura são muito semelhantes aos formatos de escrita utilizados pelo `printf`. A tabela a seguir mostra alguns formatos possíveis de leitura.

Código	Função
<code>%c</code>	Lê um char
<code>%s</code>	Lê uma string
<code>%d</code>	Lê um int
<code>%u</code>	Lê um unsigned int
<code>%hd</code>	Lê um short int
<code>%hu</code>	Lê um unsigned short int
<code>%ld</code>	Lê um long int
<code>%lu</code>	Lê um unsigned long int
<code>%f</code>	Lê um float
<code>%lf</code>	Lê um double

## Cuidado com a leitura de caracteres

Para garantir que o símbolo %c não leia um espaço em branco, nem um símbolo de tabulação (\t) e nem uma quebra de linha (\n), deve-se usar um espaço em branco antes do símbolo %c. Exemplo:

```
#include <stdio.h>
```

```
int main() {  
    char c;  
    int i;  
    printf("Digite um numero inteiro: ");  
    scanf("%d", &i);  
    printf("Digite um caractere: ");  
    scanf(" %c", &c);          /* note o espaco antes do %c */  
    printf("Os valores digitados foram '%c' e '%d'\n", c, i);  
    return 0;  
}
```

# Expressões

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

## Exemplo:

$a + b$

Calcula a soma de  $a$  e  $b$

# Expressões aritméticas

- Os operadores aritméticos são:  $+$ ,  $-$ ,  $*$ ,  $/$  e  $\%$ .
- $\langle \text{expressão} \rangle + \langle \text{expressão} \rangle$ : Calcula a soma de duas expressões.  
Ex.:  $a + b$
- $\langle \text{expressão} \rangle - \langle \text{expressão} \rangle$ : Calcula a subtração de duas expressões.  
Ex.:  $a - b$
- $\langle \text{expressão} \rangle * \langle \text{expressão} \rangle$ : Calcula o produto de duas expressões.  
Ex.:  $a * b$

# Expressões

- $\langle \text{expressão} \rangle / \langle \text{expressão} \rangle$ : Calcula a divisão de duas expressões.  
Ex.:  $a / b$
- $\langle \text{expressão} \rangle \% \langle \text{expressão} \rangle$ : Calcula o resto da divisão (inteira) de duas expressões.  
Ex.:  $a \% b$
- $- \langle \text{expressão} \rangle$ : Inverte o sinal da expressão.  
Ex.:  $-b$

# Expressões

- As expressões aritméticas (e todas as expressões) operam sobre outras expressões.
- É possível compor expressões mais complexas como:  
$$a = b + 2 + c * (9 + d / 8);$$

Qual o valor da expressão  $5 + 10 \% 3$ ?  
E da expressão  $5 * 10 \% 3$ ?

# Precedência

- Precedência é a ordem na qual os operadores serão calculados quando o programa for executado. Em C, os operadores são calculados na seguinte ordem:
  - ▶ \* e /, na ordem em que aparecerem na expressão.
  - ▶ %
  - ▶ + e -, na ordem em que aparecerem na expressão.
- Exemplo:  $8 + 10 * 6$  é igual a 68.

## Alterando a precedência

- (`<expressão>`) também é uma expressão, que calcula o resultado da expressão dentro dela para só então permitir que as outras expressões executem.  
Ex.:  $5 + 10 \% 3$  retorna 6, enquanto  $(5 + 10) \% 3$  retorna 0.
- Você pode usar quantos parênteses desejar dentro de uma expressão, contanto que utilize o mesmo número de parênteses para abrir e fechar expressões.
- Observação: use sempre parênteses em expressões para deixar claro em qual ordem a expressão é avaliada.

# Incremento(++ ) e Decremento(-- )

- Operadores de incremento e decremento têm duas funções: servem como uma expressão e incrementam ou decrementam o valor da variável ao qual estão associados em uma unidade.  
Ex.: `a++` incrementa o valor da variável `a` em uma unidade.
- Dependendo da posição do operador de incremento e decremento, uma função é executada antes da outra.

# Incremento(++ ) e Decremento(-- )

- Operador à esquerda da variável: primeiro, a variável é incrementada, depois a expressão retorna o valor da variável. Exemplo:

```
#include <stdio.h>
```

```
int main() {  
    int a = 10;  
    printf("%d\n", ++a);  
    return 0;  
}
```

- Saída:

```
11
```

# Incremento(++ ) e Decremento(-- )

- Operador à esquerda da variável: primeiro, a variável é incrementada, depois a expressão retorna o valor da variável. Exemplo:

```
#include <stdio.h>

int main() {
    int a = 10;
    printf("%d\n", ++a);
    return 0;
}
```

- Saída:

11

# Incremento(++ ) e Decremento(-- )

- Operador à direita da variável: primeiro, a expressão retorna o valor da variável, e depois a variável é incrementada. Exemplo:

```
#include <stdio.h>
```

```
int main() {  
    int a = 10;  
    printf("%d\n", a++);  
    return 0;  
}
```

- Saída:

```
10
```

# Incremento(++ ) e Decremento(-- )

- Operador à direita da variável: primeiro, a expressão retorna o valor da variável, e depois a variável é incrementada. Exemplo:

```
#include <stdio.h>

int main() {
    int a = 10;
    printf("%d\n", a++);
    return 0;
}
```

- Saída:  
10

## Incremento(++ ) e Decremento(-- )

- Em uma expressão, os operadores de incremento e decremento são sempre calculados primeiro (têm maior precedência). Exemplo:

```
#include <stdio.h>
```

```
int main() {  
    int a = 10;  
    printf("%d\n", a * ++a);  
    return 0;  
}
```

- Saída:

```
121
```

# Incremento(++ ) e Decremento(-- )

- Em uma expressão, os operadores de incremento e decremento são sempre calculados primeiro (têm maior precedência). Exemplo:

```
#include <stdio.h>
```

```
int main() {  
    int a = 10;  
    printf("%d\n", a * ++a);  
    return 0;  
}
```

- Saída:  
121

# Incremento(++) e Decremento(--)

- Em uma expressão, os operadores de incremento e decremento são sempre calculados primeiro (têm maior precedência). Exemplo:

```
#include <stdio.h>
```

```
int main() {  
    int a = 10;  
    printf("%d\n", a * a++);  
    return 0;  
}
```

- Saída:

```
100
```

# Incremento(++ ) e Decremento(-- )

- Em uma expressão, os operadores de incremento e decremento são sempre calculados primeiro (têm maior precedência). Exemplo:

```
#include <stdio.h>
```

```
int main() {  
    int a = 10;  
    printf("%d\n", a * a++);  
    return 0;  
}
```

- Saída:  
100

# Atribuições simplificadas

Uma expressão da forma:

$$a = a + b;$$

em que ocorre uma atribuição a uma das variáveis da expressão, pode ser simplificada como:

$$a += b;$$

# Atribuições simplificadas

Comando	Exemplo	Corresponde a:
<code>+=</code>	<code>a += b;</code>	<code>a = a + b;</code>
<code>-=</code>	<code>a -= b;</code>	<code>a = a - b;</code>
<code>*=</code>	<code>a *= b;</code>	<code>a = a * b;</code>
<code>/=</code>	<code>a /= b;</code>	<code>a = a / b;</code>
<code>%=</code>	<code>a %= b;</code>	<code>a = a % b;</code>

# Conversão de tipos

- É possível converter alguns tipos entre si.
- Existem duas formas de fazer a conversão: implícita e explícita.
- Implícita:
  - ▶ Capacidade (tamanho) do destino deve ser maior que a origem, caso contrário, haverá perda de informação.  
Ex.: `int a; short int b = 5; a = b;`  
Ex.: `double a; float b = 3.2; a = b;`
- Explícita:
  - ▶ Explicitamente informa o tipo para o qual o valor da variável ou expressão é convertida.  
Ex.: `a = (int) ((float) b / (float) c);`
  - ▶ Não modifica o tipo “real” da variável, só o valor de uma expressão.  
Ex.: `int a; (float) a = 1.0;` ← Errado

## Um uso da conversão de tipos

A operação de divisão (/) possui dois modos de operação de acordo com os seus argumentos: inteira ou de ponto flutuante.

- Se os dois argumentos forem inteiros, acontece a divisão inteira. A expressão  $10 / 3$  tem como valor 3.
- Se um dos dois argumentos for de ponto flutuante, acontece a divisão de ponto flutuante. A expressão  $1.5 / 3$  tem como valor 0.5.

Quando se deseja obter o valor de ponto flutuante de uma divisão (não-exata) de dois inteiros, basta converter um deles para ponto flutuante:

### Exemplo:

A expressão  $10 / (\text{float}) 3$  tem como valor 3.33333333