# MC102 – Algoritmos e Programação de Computadores

Instituto de Computação

UNICAMP

Primeiro Semestre de 2016

# Roteiro

# Indentação

- A indentação refere-se ao espaçamento ou tabulação inserida no início das linhas no código fonte de um programa.
- Seu objetivo é indicar quais elementos pertencem a um bloco de comandos.
- Embora modifique o código apenas do ponto de vista estético, a indentação facilita a leitura e interpretação do programa.

# Indentação

## 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.
- Comentários são delimitados pelos símbolos "/*" e "*/" e podem se estender por múltiplas linhas.
- O padrão ISO C 1999 (`std=c99`) permite usar o símbolo "//" para comentários de uma única linha.
- Comentários não devem conter acentos, caso contrário, podem apresentar problemas no SuSy (devido à codificação do arquivo).
- Assim como os comandos, recomenda-se que os comentários sejam indentados.

# Comentários

## Exemplo:

```
#include <stdio.h>

/* Meu primeiro programa:
   Este programa imprime uma mensagem na saida padrao */

int main() {
  /* Imprime uma mensagem */
  printf("Hello, world!\n");

  return 0;
}
```

# Comentários

## Exemplo:

```
#include <stdio.h>

/* Meu primeiro programa:
   Este programa imprime uma mensagem na saida padrao */

int main() {
  // Imprime uma mensagem
  printf("Hello, world!\n");

  return 0;
}
```

# Comentários

## Exemplo:

```
#include <stdio.h>

// Meu primeiro programa:
// Este programa imprime uma mensagem na saida padrao

int main() {
  // Imprime uma mensagem
  printf("Hello, world!\n");

  return 0;
}
```

# Imprimindo uma mensagem

- Pode-se imprimir um texto na saída padrão 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?`

- A constante *string* pode conter comandos especiais. O caractere 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 textos simples, o conteúdo de uma variável, de uma constante ou de uma expressão, utilizando o comando `printf`.
- Para isso, utiliza-se símbolos especiais no texto para indicar que aquele trecho deve ser substituído por um valor de um certo tipo, e depois, passa-se uma lista de variáveis, constantes ou expressões, separadas por vírgulas.

**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 um valor do tipo `char`, enquanto `"%d"` deve ser substituído por um valor do tipo `int`.

# Formatos inteiros

"%d": Imprime um valor inteiro.

**Exemplo:**

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

- Saída:

  ```
  10 anos
  ```

**Exemplo:**

```
int a = 12, b = 7;
printf("Soma = %d, Diferenca = %d\n", a + b, a - b);
```

- Saída:

  ```
  Soma = 19, Diferenca = 5
  ```

# Formatos inteiros

- O formato "%d" deve ser substituído pelos formatos "%ld", "%u" e "%lu", quando se deseja imprimir valores do tipo `long int`, `unsigned int` ou `unsigned long int`, respectivamente.

**Exemplo:**

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

- Saída:
  -294967296

**Exemplo:**

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

- Saída:
  4000000000

# Formatos inteiros

- O formato "%d" deve ser substituído pelos formatos "%ld", "%u" e "%lu", quando se deseja imprimir valores do tipo `long int`, `unsigned int` ou `unsigned long int`, respectivamente.

**Exemplo:**

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

- Saída:

  1705032704

**Exemplo:**

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

- Saída:

  6000000000

# Formatos de ponto flutuante

"%f": Imprime um valor em ponto flutuante, com 6 casas decimais.

**Exemplo:**

```
printf("Saldo = R$%f\n", 10.50);
```

- Saída:

  ```
  Saldo = R$10.500000
  ```

**Exemplo:**

```
printf("Resultado = %f\n", 4 * 0.72);
```

- Saída:

  ```
  Resultado = 2.880000
  ```

# Formatos de ponto flutuante

"%.<d>f": Imprime um valor em ponto flutuante, com <d> casas decimais.

**Exemplo:**

```
printf("Saldo = R$%.2f\n", 10.50);
```

- Saída:

  ```
  Saldo = R$10.50
  ```

**Exemplo:**

```
printf("Resultado = %.0f\n", 4 * 0.72);
```

- Saída:

  ```
  Resultado = 3
  ```

# Formatos de ponto flutuante

"%e": Imprime um valor em ponto flutuante, em notação científica.

**Exemplo:**

```
printf("Valor = %e\n", 100.2545);
```

- Saída:

  ```
  Valor = 1.002545e+02
  ```

**Exemplo:**

```
printf("Valor = %e\n", 0.000724);
```

- Saída:

  ```
  Valor = 7.240000e-04
  ```

# Formato caractere

"%c": Imprime um caractere.

**Exemplo:**

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

- Saída:

  ```
  A = 65
  ```

**Exemplo:**

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

- Saída:

  ```
  e = 101
  ```

# Formato *string*

"%s": Imprime uma *string* (cadeia de caracteres).

**Exemplo:**

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

- Saída:

  ```
  Meu primeiro programa.
  ```

**Exemplo:**

```
printf("%s, %s!\n", "Hello", "world");
```

- Saída:

  ```
  Hello, world!
  ```

# A função `scanf`

- Realiza a leitura de valores a partir da entrada padrão.
- Parâmetros:
  - Uma *string*, indicando os tipos das variáveis que serão lidas e o formato dessa leitura (Exemplos: `"%d"`, `"%f"`, `"%c"`, etc).
  - Uma lista de variáveis, cada uma delas precedidas pelo caractere `'&'` (Exemplos: `&idade`, `&valor`, `&letra`, etc).
- Aguarda que o usuário forneça um valor e atribui o valor à variável.

# A função scanf

O programa abaixo é composto de quatro passos:

1. Criar uma variável para armazenar um número
2. Imprimir uma mensagem solicitando um número
3. Ler e armazenar o valor de um número
4. Imprimir o número armazenado

```
#include <stdio.h>

int main() {
  int n;
  printf("Entre com um numero: ");
  scanf("%d", &n);
  printf("O valor fornecido foi %d\n", n);
  return 0;
}
```

# Leitura de múltiplos valores

```
#include <stdio.h>

int main() {
  int x, y, z;

  printf("Entre com 3 numeros: ");
  scanf("%d %d %d", &x, &y, &z);

  printf("Valores fornecidos: %d, %d e %d\n", x, y, z);

  return 0;
}
```

# Leitura de múltiplos valores separados por vírgulas

```
#include <stdio.h>

int main() {
  int x, y, z;

  printf("Entre com 3 numeros (separados por virgulas): ");
  scanf("%d,%d,%d", &x, &y, &z);

  printf("Valores fornecidos: %d, %d e %d\n", x, y, z);

  return 0;
}
```

# Formatos de leitura de variável

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

| Formato | Função |
|---|---|
| `"%d"` | Ler um valor do tipo `int` |
| `"%u"` | Ler um valor do tipo `unsigned int` |
| `"%hd"` | Ler um valor do tipo `short int` |
| `"%hu"` | Ler um valor do tipo `unsigned short int` |
| `"%ld"` | Ler um valor do tipo `long int` |
| `"%lu"` | Ler um valor do tipo `unsigned long int` |
| `"%f"` | Ler um valor do tipo `float` |
| `"%lf"` | Ler um valor do tipo `double` |
| `"%c"` | Ler um valor do tipo `char` |
| `"%s"` | Ler uma *string* (cadeia de caracteres) |

# Cuidado com a leitura de caracteres

Para garantir que o formato "%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 (" %c"). Exemplo:

```
#include <stdio.h>

int main() {
  char c;
  int i;

  printf("Entre com um numero inteiro: ");
  scanf("%d", &i);

  printf("Entre com um caractere: ");
  scanf(" %c", &c);

  printf("Os valores fornecidos foram '%c' e '%d'.\n", c, i);
  return 0;
}
```

# Expressões aritméticas

- Já vimos que constantes e variáveis são consideradas expressões.
- Uma expressão pode conter um conjunto de operações aritméticas, lógicas ou relacionais.
- Os operadores aritméticos são: `+`, `-`, `*`, `/` e `%`.

# Expressões aritméticas

- `<expressão> + <expressão>`: calcula a soma de duas expressões.
  Exemplo: `a + b`
- `<expressão> - <expressão>`: calcula a subtração de duas expressões.
  Exemplo: `a - b`
- `<expressão> * <expressão>`: calcula o produto de duas expressões.
  Exemplo: `a * b`
- `<expressão> / <expressão>`: calcula a divisão de duas expressões.
  Exemplo: `a / b`
- `<expressão> % <expressão>`: calcula o resto da divisão (inteira) de duas expressões.
  Exemplo: `a % b`
- `-<expressão>`: inverte o sinal da expressão.
  Exemplo: `-a`

# Expressões aritméticas

- As expressões aritméticas (e todas as demais expressões) operam sobre outras expressões.
- É possível compor expressões mais complexas.
  Exemplo: `a + 2 + b * 9 - c / 8`
- Qual o valor das seguintes expressões?
  - `5 + 10 % 3`
  - `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.
- Exemplos:
  - `5 + 10 % 3` é igual a 6.
  - `5 * 10 % 3` é igual a 2.

# Alterando a precedência

- `(<expressão>)` também é uma expressão, que indica que o resultado da expressão interna deve ser calculado antes de se permitir que outras expressões executem sobre ela.
- Exemplos:
  - `5 + 10 % 3` é igual a 6.
  - `(5 + 10) % 3` é igual a 0.
- Você pode usar quantos parênteses desejar dentro de uma expressão, contanto que todos os pares de parênteses sejam balanceados, ou seja, para cada `'('` exista um `')'` correspondente.
- Observação: em expressões mais complexas, sempre use parênteses para deixar claro em qual ordem as expressões devem ser avaliadas.

# Incremento (++) e decremento (−−)

- Operadores de incremento e decremento têm duas funções:
  - Incrementam ou decrementam em uma unidade o valor da variável à qual estão associados.
  - Podem ser usados como uma expressão.
- Exemplos:
  - `x++` ou `++x` incrementam o valor da variável `x` em uma unidade.
  - `x--` ou `--x` decrementam o valor da variável `x` em uma unidade.
- Dependendo da posição do operador de incremento ou de decremento em relação à variável, uma função é executada antes da outra.

# Incremento (++) e decremento (--)

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

```
#include <stdio.h>

int main() {
  int a = 10;

  printf("%d\n", ++a);
  printf("%d\n", a);

  return 0;
}
```

- Saída:

```
11
11
```

# Incremento (++) e decremento (--)

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

```
#include <stdio.h>

int main() {
  int a = 10;

  printf("%d\n", a++);
  printf("%d\n", a);

  return 0;
}
```

- Saída:

```
10
11
```

# Incremento (++) e decremento (−−)

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

```
#include <stdio.h>

int main() {
  int a = 10;

  printf("%d\n", 2 * ++a);
  printf("%d\n", a);

  return 0;
}
```

- Saída:

```
22
11
```

# Incremento (++) e decremento (−−)

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

```
#include <stdio.h>

int main() {
  int a = 10;

  printf("%d\n", 2 * a++);
  printf("%d\n", a);

  return 0;
}
```

- Saída:

```
20
11
```

# Atribuições simplificadas

Uma atribuição da forma:

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

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

```
a += b;
```

# Atribuições simplificadas

| Operador | Exemplo | Correspondente |
|---|---|---|
| `+=` | `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;` |

# Atribuições simplificadas

As seguintes atribuições (incremento de uma variável) são equivalentes:

- `a = a + 1;`
- `a += 1;`
- `a++;`
- `++a;`

# Conversão de valores de tipos diferentes

- É possível converter valores de alguns tipos em outros tipos.
- Existem duas formas de converter valores de tipos diferentes:
  - Conversão implícita
  - Conversão explícita

# Conversão implícita

- A capacidade (tamanho) do destino deve ser maior do que a da origem, caso contrário, poderá haver perda de informação.
- Exemplo sem perda de informação:

```
int a;
short int b = 5;
a = b;
```

- Exemplo sem perda de informação:

```
double a;
float b = 3.2;
a = b;
```

# Conversão implícita

- Exemplo com perda de informação:

```
short int a, b;
int x = 2016, y = 123456;
a = x;                          /* a = 2016  */
b = y;                          /* b = -7616 */
```

- Exemplo com perda de informação:

```
float a, b;
double x = 12.34, y = 2e50;
a = x;                          /* a = 12.34 */
b = y;                          /* b = inf   */
```

# Conversão implícita

- Exemplo com perda de informação:

```
unsigned int a, b;
int x = 100, y = -128;
a = x;                          /* a = 100         */
b = y;                          /* b = 4294967168 */
```

- Exemplo com perda de informação:

```
int a, b;
double x = 3.2, y = -1.95;
a = x;                          /* a = 3  */
b = y;                          /* b = -1 */
```

# Conversão explícita

- É possível explicitamente informar o tipo para o qual o valor deve ser convertido, usando a seguinte notação:

```
(tipo) valor
```

  - Exemplo:

```
float a;
int b = 23, c = 4;
a = (float) b / (float) c;
```

- Importante: não é possível modificar o tipo de uma variável, apenas converter o tipo de expressão.

  - Exemplo:

```
int a;
/* Erro: impossivel converter o tipo de uma variavel */
(float) a = 1.5;
```

# Divisão inteira × divisão de ponto flutuante

- 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. Por exemplo, a expressão `10 / 3` tem como valor 3.
- Se um dos dois argumentos for do tipo ponto flutuante, acontece a divisão de ponto flutuante. Por exemplo, a expressão `10.0 / 3`, assim como a expressão `10 / 3.0` tem como valor `3.333333`.

# Divisão inteira × divisão de ponto flutuante

- Quando se deseja obter o valor de ponto flutuante de uma divisão de dois inteiros, deve-se converter, explicitamente, o valor de pelo menos um deles para ponto flutuante.
  - Exemplo:

```
int x = 9, y = 4;
float z;
z = x / y;              /* z = 2.000000 */
z = x / (float) y;      /* z = 2.250000 */
z = (float) x / y;      /* z = 2.250000 */
z = (float) (x / y);    /* z = 2.000000 */
```

# Biblioteca matemática

- A biblioteca `math.h` fornece uma série de funções matemáticas pré-definidas.
- Para usá-la, deve-se:
  - Incluir a biblioteca, no início do programa, usando o comando:
    `#include <math.h>`
  - Compilar o programa usando a opção `-lm`:
    `gcc -lm teste.c -o teste`

# Funções da biblioteca matemática

- `abs(x)`: calcula o valor absoluto de um inteiro `x`.
- `sqrt(x)`: calcula a raiz quadrada de `x`.
- `pow(x, y)`: calcula o valor de $x^y$.
- `log(x)`: calcula o logaritmo natural (base `e`) de `x`.
- `exp(x)`: calcula o valor de $e^x$.
- `sin(x)`: calcula o seno de `x` (`x` em radianos).
- `cos(x)`: calcula o cosseno de `x` (`x` em radianos).
- `tan(x)`: calcula a tangente de `x` (`x` em radianos).
- ... e muitas outras.

# Exemplo - Cálculo da hipotenusa de um triângulo retângulo

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

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

  printf("Entre com os valores dos catetos: ");
  scanf("%lf %lf", &a, &b);

  c = sqrt(pow(a,2) + pow(b,2));
  printf("Hipotenusa: %.2f\n", c);

  return 0;
}
```

# Exercícios

- Escreva um programa que, dada uma função do segundo grau $f(x) = ax^2 + bx + c$, representada pelos seus coeficientes $a$, $b$ e $c$, calcule e imprima o valor de $f(x)$, para um número $x$ também dado.
- Escreva um programa que, dados dois números positivos, calcule e imprima as médias aritmética, geométrica e harmônica entre eles.
- Escreva um programa que, dadas as coordenadas de dois pontos $(x_1, y_1, z_1)$ e $(x_2, y_2, z_2)$, calcule e imprima a distância euclidiana entre eles.
- Escreva um programa que, dados dois horários $H_1$ e $H_2$, ambos no mesmo dia e no formato `HH:MM`, tais que $H_1 \leq H_2$, calcule e imprima o número de minutos decorridos entre $H_1$ e $H_2$.
- Escreva um programa que, dadas duas retas concorrentes $r_1$ e $r_2$, ambas também concorrentes aos eixos $x$ e $y$ e representadas pelas suas equações de reta no formato $ax + by + c = 0$, determine e imprima o ponto em que $r_1$ e $r_2$ se cruzam.