

# SI100

## Algoritmos e Programação de Computadores I

1º Semestre - 2018

### Tópico 2b:

# Linguagem Estruturada e Introdução à Linguagem C

Unidades 5 e 6

Prof. Guilherme Palermo Coelho  
[guilherme@ft.unicamp.br](mailto:guilherme@ft.unicamp.br)

# Roteiro

- Aspectos Fundamentais da Linguagem C:
  - A função *printf()*;
  - A função *scanf()*;
  - Operadores;
- Exercícios;
- Referências.

# ASPECTOS FUNDAMENTAIS DA LINGUAGEM C

# A FUNÇÃO PRINTF()

# A função printf()

- Como já vimos, a função ***printf()*** permite, dentre outras coisas, imprimir informações na tela;
  - Faz parte da biblioteca padrão de C (stdio.h);
  - Pode receber um ou mais parâmetros, separados por vírgulas;
  - Permite o uso de ***caracteres especiais***;
    - Ex.: \n (insere uma quebra de linha)
    - Também chamados de ***caracteres de escape***;
- Permite o uso de ***códigos para impressão formatada*** (códigos que são substituídos por valores com uma dada formatação);
  - Ex.: %d (é substituído por um valor inteiro).

# A função printf()

- Caracteres especiais da função ***printf()***:

Caractere	Significado
\a	Caractere de alerta (emite som no computador)
\b	Retrocesso ( <i>backspace</i> )
\f	Alimentação de formulário
\n	Nova linha
\r	Retorno de carro (volta para o início da linha)
\t	Tabulação horizontal
\v	Tabulação vertical
\	Barra invertida
\?	Ponto de interrogação
\'	Aspas simples (apóstrofo)
\"	Aspas duplas

# A função `printf()`

- Códigos de impressão formatada `printf()`:

Código	Formato
%c	Caractere simples
%d	Número inteiro com sinal
%e	Número em notação científica
%f	Número em ponto flutuante
%o	Número octal (base 8)
%x	Número hexadecimal (base 16)
%s	Cadeia de caracteres ( <i>string</i> )
%u	Número inteiro sem sinal
%ld	Número inteiro longo com sinal
%lf	Número em ponto flutuante longo ( <i>double</i> )

# A função printf()

- Através dos códigos de impressão formatada, é possível definir também:
  - O **tamanho mínimo** para impressão de um campo;
  - A **precisão e o arredondamento** em campos de ponto flutuante;
  - O **alinhamento** à direita ou à esquerda;
  - A complementação com **zeros à esquerda**.

# A função printf()

```
#include <stdio.h>
int main(){
    printf("Os alunos sao %d.\n",350);
    printf("Os alunos sao %6d.\n",350);
    printf("Os alunos sao %06d.\n",350);
    printf("%f\n%f\n", 3456.78, 6.78);
    printf("%10.1f\n%10.1f\n", 3456.78, 6.78);
    printf("%-10.3f\n%-10.3f\n", 3456.78, 6.78);
    return 0; }
```

```
Os alunos sao 350.
Os alunos sao      350.
Os alunos sao 000350.
3456.780000
6.780000
 3456.8
    6.8
3456.780
6.780
```

# A função printf() - caracteres

- Em C, um caractere pode ser representado de diversas maneiras:
  - O próprio caractere, entre *aspas simples*; ou
  - Sua representação numérica (código ASCII) em decimal (base 10), hexadecimal (base 16) ou octal (base 8);
    - **Valor hexadecimal:** sempre começa com 0x → Ex.: 0x41
    - **Valor octal:** sempre começa com 0 → Ex.: 0701
- O código ASCII (*American Standard Code for Information Interchange*) é um padrão internacional que associa um código numérico a cada caractere:
  - Codifica 256 caracteres diferentes;
  - <http://www.asciitable.com/>

# A função printf() - caracteres

- Exemplo:

```
#include <stdio.h>
int main(){
    printf("%d %c %x %o\n", 'A' , 'A' , 'A' , 'A');
    printf("%c %c %c %c\n", 'A' , 65 , 0x41 , 0101);
    return 0;
}
```

- Saída:

```
65 A 41 101
A A A A
```

# A FUNÇÃO SCANF()

# A função `scanf()`

- A função `scanf()` é uma função de leitura:
  - Permite a *leitura* de dados formatados da entrada padrão (teclado);
  - Aguarda a digitação dos dados pelo usuário e atribui o valor digitado a uma variável;

```
#include <stdio.h>
int main(){
    int n;
    printf("Digite um número e pressione Enter: ");
    scanf("%d", &n);
    printf("O valor digitado foi %d\n", n);
    return 0;
}
```

# A função `scanf()`

- A função `scanf()` prevê os seguintes parâmetros:
  - Uma **expressão de controle** (no formato de cadeia de caracteres);
  - Usando os mesmos **códigos de formatação** de `printf()`;

Código	Formato
%c	Caractere simples
%d	Número inteiro com sinal
%e	Número em notação científica
%f	Número em ponto flutuante
%o	Número octal (base 8)
%x	Número hexadecimal (base 16)
%s	Cadeia de caracteres ( <i>string</i> )
%u	Número inteiro sem sinal
%ld	Número inteiro longo com sinal
%lf	Número em ponto flutuante longo ( <i>double</i> )

# A função `scanf()`

- A função `scanf()` prevê os seguintes parâmetros:
  - Uma **expressão de controle** (no formato de cadeia de caracteres);
    - Usando os mesmos **códigos de formatação** de `printf()`;
  - Um ou mais argumentos, separados por vírgula;
    - Estes argumentos correspondem aos **endereços das variáveis** onde serão salvos os dados;

Endereço de uma variável = endereço da posição de memória a ela associada.

Endereço de uma variável: dado pelo operador **& (address-of)**.

**Exemplo:** `scanf ("%d", &n) ;`

# O operador *address-of* (&)

- Normalmente, os **endereços** das posições de memória em que serão armazenadas as variáveis não é conhecido no momento da escrita do programa;
  - O endereço depende do *sistema operacional* e também do *compilador* utilizado;
  - O endereço **pode mudar** em duas execuções do programa feitas na mesma máquina;
- É útil para o programador ter operadores que permitam **obter** o endereço corrente de uma dada variável;
  - Em C isto é possível através do uso do operador **&** (*address-of*);
  - **Ex.:** se **n** é uma variável do tipo inteiro, **&n** é um *endereço que referencia* a variável **n**.

# O operador *address-of* (&)

- Exemplo:

```
#include <stdio.h>

int main(){
    int n = 8;
    printf("Valor: %d, Endereco: %u\n", n, &n);

    return 0;
}
```

- Saída (o endereço pode mudar a cada execução):

Valor: 8, Endereco: 3221217828

# Exemplo: *scanf()* e *address-of (&)*

- Exemplo:

```
#include <stdio.h>

int main(){
    float anos, dias;
    printf("Digite sua idade em anos: ");
    scanf("%f",&anos);
    dias = anos * 365;
    printf("Sua idade em dias é %.0f.\n", dias);

    return 0;
}
```

- Saída:

```
Digite sua idade em anos: 22.13
Sua idade em dias é 8077.
```

# Outras funções de entrada e saída (E/S)

- Tarefa para casa:
  - Estude o funcionamento das seguintes funções de E/S da linguagem C:
    - *getchar();*
    - *putchar();*
  - Entenda como cada uma funciona, quais são as diferenças entre elas e o que foi visto anteriormente e implemente alguns pequenos programas de teste.

# OPERADORES

# Operadores aritméticos

- Como o próprio nome sugere, *operadores aritméticos* são responsáveis pela execução de *operações aritméticas* entre constantes e variáveis;
- A linguagem C oferece 06 operadores aritméticos binários (que operam sobre dois operandos) e 01 operador aritmético unário (opera sobre um único operando):

## Binários

=	Atribuição
+	Soma
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto da divisão inteira)

## Unário

-	Menos unário
---	--------------

# Operadores aritméticos – exemplo

```
#include <stdio.h>

int main(){
    int a, b, c;
    a = 4;
    b = 3;
    c = (a * b)/2;
    printf("Variavel \'a\' tem valor: %d\n", a);
    printf("Variavel \'b\' tem valor: %d\n", b);
    printf("Variavel \'c\' tem valor: %d\n", c);
    printf("Resto da div. entre a e b eh: %d\n", a % b);
    printf("O valor de '-b' eh: %d\n", -b);
    return 0; }
```

```
Variavel 'a' tem valor: 4
Variavel 'b' tem valor: 3
Variavel 'c' tem valor: 6
Resto da div. entre a e b eh: 1
O valor de '-b' eh: -3
```

# Operadores de incremento/decremento

- C possui também dois operadores **unários** que permitem *incrementar* ou *decrementar* seus operandos:
  - Operador de incremento: `++`;
  - Operador de decremento: `--`;
- Estes operadores podem funcionar em **dois** modos:
  - **Pré-fixado**: são colocados *antes* do operando;
    - **Ex.**: `++n` (*neste caso a variável n é incrementada antes do seu valor ser usado*);
  - **Pós-fixado**: são colocados *depois* do operando;
    - **Ex.**: `n--` (*neste caso a variável n é decrementada depois do seu valor ter sido usado*);

# Operadores de incremento/decremento

```
#include <stdio.h>

int main(){
    int a, b;
    b = 3;

    a = b++;
    printf("Pos-incremento: \'a\' tem valor: %d\n", a);
    printf("Variavel \'b\' tem valor: %d\n", b);

    a = ++b;
    printf("Pre-incremento: \'a\' tem valor: %d\n", a);
    printf("Variavel \'b\' tem valor: %d\n", b);

    return 0;
}
```

Pos-incremento: 'a' tem valor: 3

Variavel 'b' tem valor: 4

Pre-incremento: 'a' tem valor: 5

Variavel 'b' tem valor: 5

# Operadores de incremento/decremento

- No exemplo anterior temos que:
  - A instrução “`a = b++;`” pode ser decomposta nas seguintes instruções:
    - `a = b;`
    - `b = b + 1;`
  - A instrução “`a = ++b;`” pode ser decomposta nas seguintes instruções:
    - `b = b + 1;`
    - `a = b;`
- Os resultados com o operador de decremento (`--`) são análogos.

**Incremento/decremento só podem ser usados com variáveis!**

# Operadores aritméticos de atribuição

- Estes operadores são usados com um **nome de variável** à esquerda e uma **expressão** à direita;
  - A variável recebe o **resultado** da aplicação do operador a **ela** e à **expressão**.

- **Ex.:**

- $i += 2;$                        $\rightarrow$                $i = i + 2;$
- $x *= y + 1;$                        $\rightarrow$                $x = x * (y+1);$
- $t /= 2.5;$                        $\rightarrow$                $t = t / 2.5;$
- $p %= 5;$                        $\rightarrow$                $p = p \% 5;$
- $d -= 3;$                        $\rightarrow$                $d = d - 3;$

# Operadores aritméticos de atribuição

```
#include <stdio.h>

int main(){
    int total = 0;
    int cont = 10;;

    printf("Total=%d\n", total);
    total += cont;
    printf("Total=%d\n", total);
    total += cont;
    printf("Total=%d\n", total);

    return 0;
}
```

Total=0  
Total=10  
Total=20

# Precedência de operadores aritméticos

- Uma expressão aritmética em C como  $a+b*c$  será avaliada como  $a + (b*c)$ , pois existe uma *precedência entre os operadores*;
- Esta precedência é definida na seguinte ordem:
  - \*, / (maior precedência)
  - %
  - +, - (menor precedência)
- Operadores de **incremento/decremento** têm precedência maior que operadores aritméticos:
  - Ex.:  $a*++b$  é avaliado como  $(a)*(++b)$ ;

# Precedência de operadores aritméticos

```
#include <stdio.h>

int main() {
    int a = 2;
    int b = 3;
    int c = a-++b*3;

    printf("c=%d\n", c);
    printf("b=%d\n", b);

    return 0;
}
```

c=-10

b=4

# Operadores relacionais

- Operadores *relacionais* são usados para *fazer comparações*;
  - São eles:

Operador	Significado
>	maior
$\geq$	maior igual
<	menor
$\leq$	menor igual
$=\!=$	igualdade
$!=$	diferente

Cuidado para não usar “=” ao invés de “==”!

- Operadores relacionais retornam valores inteiros com significado lógico:
  - 1 = “verdadeiro”;
  - 0 = “falso”;

# Operadores relacionais

```
#include <stdio.h>

int main() {
    int verdad, falso;
    verdad = ( 15 < 20 );
    falso = ( 15 == 20 );
    printf("Verdadeiro = %d, falso = %d\n",verdad, falso);

    return 0;
}
```

Verdadeiro = 1, falso = 0

# Precedência de operadores relacionais

- A *precedência* de um **operador relacional** é mais baixa que a dos operadores aritméticos;
- Ou seja, eles são avaliados *após* os operadores aritméticos;
- Ex.:
  - $3+5==5$  é avaliado como  $(3+5) == 5$

# EXERCÍCIOS

# Exercícios

1. Segundo o plano da disciplina SI100, a média final de um aluno é dada por:  $(P1 + P2 + T)/3$ . Implemente um programa que leia as notas de P1, P2 e T de um dado aluno e imprima sua média final.  
**Atenção:** cuidado para que a divisão por 3 seja feita na **soma das notas**, e não apenas em um dos termos.
2. Faça um programa que leia um caractere do teclado e imprima o caractere seguinte (posterior) na tela. **Dica:** veja o que acontece ao se adicionar um valor numérico inteiro a uma variável do tipo caractere.
  - Ex.: entrada = a, saída = b
3. Escreva um programa que recebe como entrada uma temperatura (número real) em graus **Celsius** e a transforme para **Farenheit**. Lembre-se que  $T_{\text{Farenheit}} = T_{\text{Celsius}} * 1.8 + 32$ .

# Exercícios

4. Faça o inverso do exercício 10, ou seja, escreva um programa que recebe como entrada uma temperatura (número real) em graus **Farenheit** e a transforme para **Celsius**.
5. Elabore um programa que leia do teclado a **idade**, a **altura** e o **sexo** de uma pessoa e escreva uma frase apresentando estes dados.
  - Ex.:

Entre com dados (idade, altura, turma): 51

1.95

F

A pessoa tem 51 anos, 1.95 de altura e é do sexo F.

# Exercícios

6. Escreva um programa em C que receba a idade do usuário em *anos* e imprima na tela sua idade em *minutos* (desconsidere a existência de anos bissextos).

# REFERÊNCIAS

# Referências

- MIZRAHI, V. V., *Treinamento em Linguagem C – Curso Completo*, 2a Edição, Pearson Makron Books, 2005.
- MIYAZAWA, F. K., DE SOUZA, C. C. & KOWALTOWSKI, T.. *Notas de Aula da disciplina Algoritmos e Programação de Computadores*. Instituto de Computação, Unicamp.