

Linguagem C

Prof. Roberto Hugo Wanderley Pinheiro

roberto.hugo@ufca.edu.br



**UNIVERSIDADE
FEDERAL DO CARIRI**

Motivação

- Ver todo o conteúdo do semestre em uma única aula de revisão gigante e passar o semestre inteiro apenas aplicando, praticando, programando e sofrendo

Motivação

- Ver todo o conteúdo do semestre em uma única aula de revisão gigante e passar o semestre inteiro apenas aplicando, praticando, programando e sofrendo



Motivação

- Ver todo o conteúdo do semestre em uma única aula de revisão gigante e passar o semestre inteiro apenas aplicando, praticando, programando e sofrendo
- Brincadeirinha...



Motivação

- Ver todo o conteúdo do semestre em uma única aula de revisão gigante e passar o semestre inteiro apenas aplicando, praticando, programando e sofrendo
- Brincadeirinha...
- Sempre que necessário podemos rever algum assunto por completo mas o ideal é focar na prática

Motivação

- Ver todo o conteúdo do semestre em uma única aula de revisão gigante e passar o semestre inteiro apenas aplicando, praticando, programando e sofrendo
- Brincadeirinha...
- Sempre que necessário podemos rever algum assunto por completo mas o ideal é focar na prática
- Não é bem uma motivação, mas é isso aí



Variáveis

- Declaração de uma variável
 <tipo da variável> <nome da variável>;
 - Exemplo: float valor1;

Variáveis

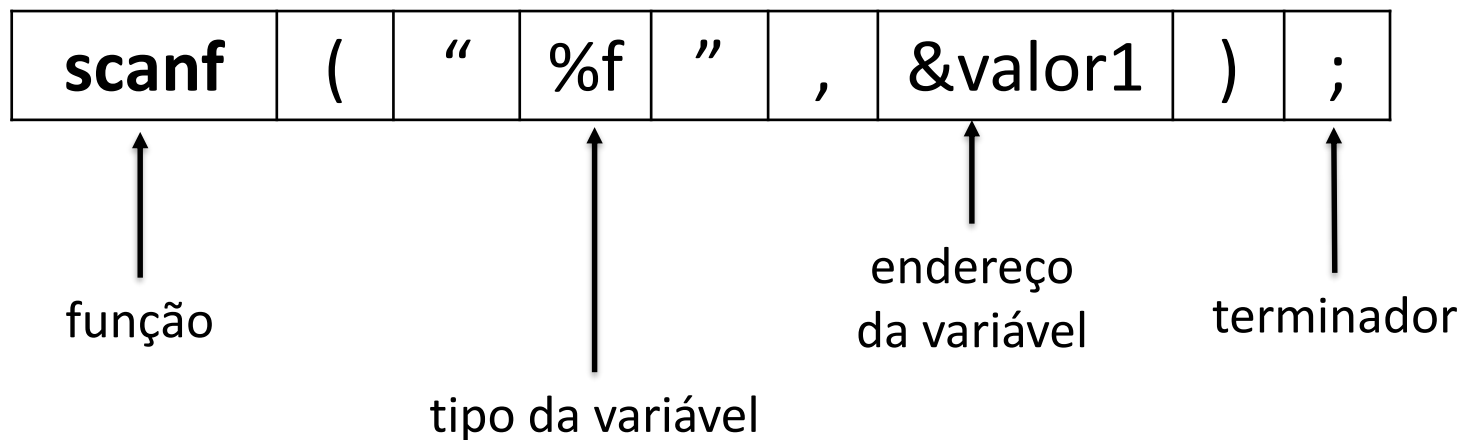
- Regras para nomear uma variável
 - Iniciar com letra
 - Não possuir caracteres especiais (@, #, %)
 - Não possui acentos (ã, ú, â)
 - Não conter espaços (_ pode ser usado)
 - Não pode ser uma palavra reservada

Variáveis

- Tipos de dados em C

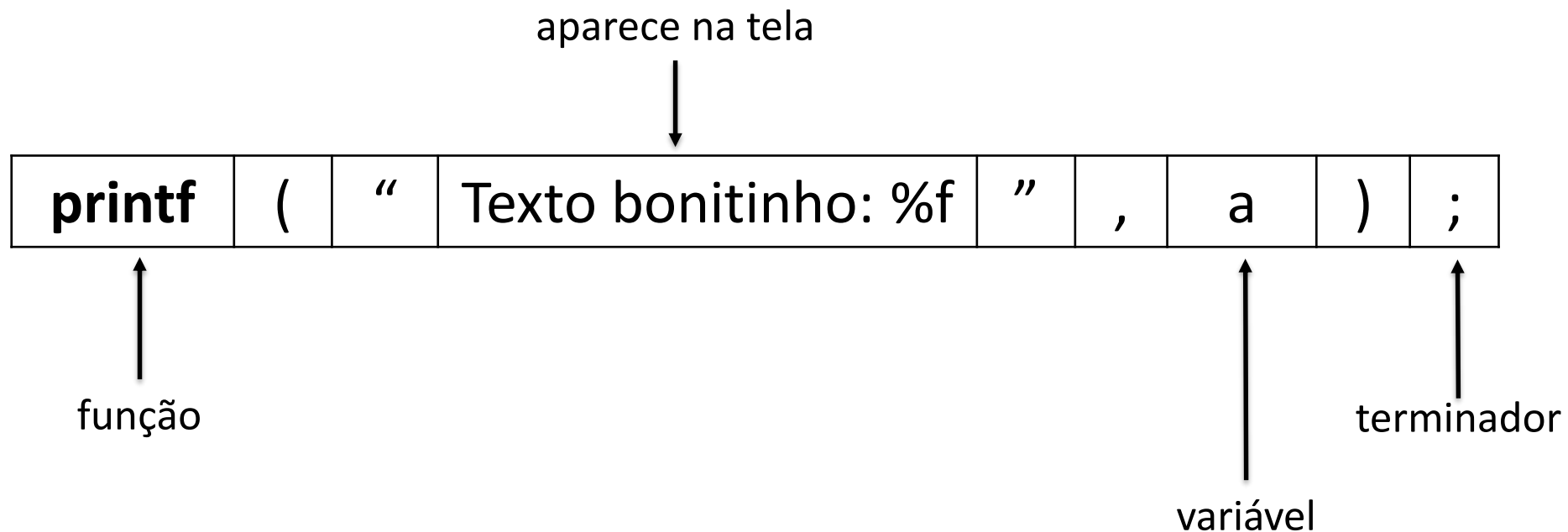
Descrição	Tipo	Valores Válidos	Memória
Inteiro pequeno	short int	-32767 a +32767	2 bytes
Inteiro	int	-2147483647 a +2147483647	4 bytes
Inteiro com mais dígitos	long int	-uma penca a +um moi	8 bytes
Caractere	char	Caractere em ASCII	1 byte
Real	float	2^{-37} a 2^{+37}	4 bytes
Real com mais casas decimais	double	2^{-37} a 2^{+37}	8 bytes

scanf



Tipo	%?
int	%d
long long int	%lld
char	%c
float	%f
double	%lf

printf



Operadores de Atribuição

Operador	Descrição	Exemplo
=	Atribuição	<code>valor = 10</code> <code>pi = 3.14</code>
+=	Atribuição com soma, subtração, multiplicação e divisão, respectivamente	<code>a += 1</code>
-=		<code>a -= 2</code>
*=		<code>a *= 5</code>
/=		<code>a /= 2</code>
++	Atribuição com soma de exatamente um	<code>a++</code>
--	Atribuição com subtração de exatamente um	<code>a--</code>

Operadores Aritméticos

Operador	Descrição	Exemplo
+	Adição	<code>x = 5 + 4; // x = 9</code>
-	Subtração	<code>x = 5 - 4; // x = 1</code>
*	Multiplicação	<code>x = 5 * 4 // x = 20</code>
/	Divisão	<code>x = 5.0 / 4.0; // x = 1.25</code>
%	Resto da Divisão	<code>x = 10 % 3; // x = 1</code>

Operadores Relacionais

Operador	Descrição	Exemplo
>	Maior?	5 > 4 // Saída True
>=	Maior ou igual?	5 >= 4 // Saída True
<	Menor?	5 < 4 // Saída False
<=	Menor ou igual?	5 <= 4 // Saída False
==	Igual?	5 == 4 // Saída False
!=	Diferente?	5 != 4 // Saída True

Operadores Lógicos

Operador	Descrição	Exemplo
&&	E	<code>5 == 4 && 5 > 4 // Saída: False</code>
	OU	<code>5 == 4 5 > 4 // Saída: True</code>
!	NÃO	<code>!(5 == 4 5 > 4) // Saída: False</code>

Instrução if

- Sintaxe

```
if (<condição>) {  
    <comando_do_if1>  
    <comando_do_if2>  
    ...  
}  
<comando_fora_do_if>
```

Se a condição for **True**:
todos esses comandos
serão executados.

Se a condição for **False**:
todos os comandos dentro
do if serão desconsiderados.

Instrução if else

- Sintaxe

```
if (<condição>) {  
    <comando_do_if1>  
    <comando_do_if2>  
    ...  
}  
else {  
    <comando_do_else1>  
    <comando_do_else2>  
    ...  
}  
<comando_fora_do_if>
```

Se a condição for **True**:
todos esses comandos
serão executados

Se a condição for **False**:
todos esses comandos
serão executados

Estruturas Condicionais aninhadas

```
#include <stdio.h>

int main () {
    float mp;
    printf("Digite sua MP: ");
    scanf("%f", &mp);
    if (mp >= 9.0)
        printf("Conceito A");
    else if (mp >= 8.0)
        printf("Conceito B");
    else if (mp >= 7.0)
        printf("Conceito C");
    else if (mp >= 4.0)
        printf("Conceito E");
    else
        printf("Conceito F");
    return 0;
}
```

Instrução switch

- Sintaxe

```
switch (<variável>) {  
    case <valor1>:  
        <comando1>  
        <comando2>  
        ...  
        break;  
    case <valor2>:  
        ...  
    ...  
    default:  
        <comando1>  
        <comando2>  
        ...  
}  
<comando_fora_do_switch>
```

Se a variável tiver o valor igual ao <valor1>, os comandos daqui são executados


Se a variável tiver um valor não contemplado por nenhum dos casos, os comandos daqui são executados

Instrução switch

- Sintaxe

```
switch (<variável>) {  
    case <valor1>:  
        <comando1>  
        <comando2>  
        ...  
        break;  
    case <valor2>:  
        ...  
    ...  
    default:  
        <comando1>  
        <comando2>  
        ...  
}  
<comando_fora_do_switch>
```

O comando **break** faz com que um Bloco de Comandos seja finalizado imediatamente, indo diretamente para cá



Instrução **while**

- Sintaxe

```
while (<condição>) {  
    <comando_do_while1>  
    <comando_do_while2>  
    .  
    .  
    .  
}  
<comando_fora_do_while>
```

Enquanto a
condição for **True**:
executa todos esses
comandos
Quando a condição
for **False**:
encerra o laço

Instrução do-while

- Sintaxe

```
do {  
    <comando_do_dowhile1>  
    <comando_do_dowhile2>  
    .  
    .  
    .  
} while (<condição>);  
<comando_fora_do_dowhile>
```

Executa **sempre** uma vez e depois depende da condição
Enquanto a condição for **True**:
 retorna e executa todos esses comandos
Quando a condição for **False**:
 encerra o laço

Instrução **while**

- Comparação **do-while** x **while** para validação

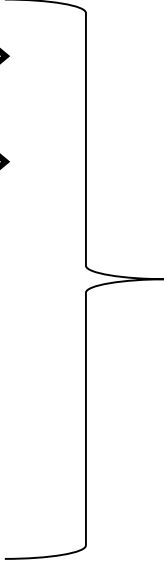
```
int n;  
do {  
    printf("Digite um número positivo: ");  
    scanf("%d", &n);  
} while (n <= 0);  
printf("Número válido!");
```

```
int n;  
printf("Digite um número positivo: ");  
scanf("%d", &n);  
while (n <= 0) {  
    printf("Número inválido!\n");  
    printf("Digite um número positivo: ");  
    scanf("%d", &n);  
}  
printf("Número válido!");
```

Instrução for

- Sintaxe

```
for (<inicio>;<parada>;<passo>) {  
    <comando_do_for1>  
    <comando_do_for2>  
    .  
    .  
    .  
}  
<comando_fora_do_for>
```



Comandos serão executados até que a condição de parada seja **False**.

Declarar Vetor

```
int idades[10];
```

```
float notas[] = {5.4, 7.8};
```

Colchetes

Usaremos para indicar tamanho e posição

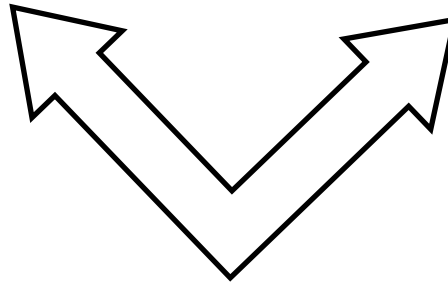
Chaves

Usaremos para armazenamento inicial apenas

Acessar dados do Vetor

```
float notas[] = {5.8, 7.8}
```

```
media = (notas[0]+notas[1])/2.0
```



Índices

Variam de 0 até
tamanho do vetor - 1

Atenção: cuidado para não acessar índice inexistente

Preencher Vetor

```
float notas[30];  
int i;  
  
for (i = 0 ; i < 30 ; i++)  
    scanf("%f", &notas[i]);
```

Usar String

- Declarar

```
char nome[30];
```

ou

```
char nome[30] = "Roberto Pinheiro";
```

- Preencher

```
gets(nome);
```

- Imprimir

```
printf("%s", nome);
```

Funções de String

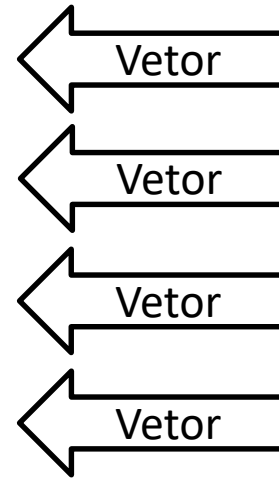
Função	Descrição
strcpy(s1,s2)	Copia s2 em s1
strcat(s1,s2)	Concatena s2 ao final de s1
strlen(s1)	Retorna o tamanho de s1
strcmp(s1,s2)	Retorna 0 se iguais; negativo se $s1 < s2$ e positivo c.c.
atoi(s1)	Converte o texto em inteiro
atof(s1)	Converte o texto em float

Funções disponíveis com
`#include <string.h>`

Exceto as conversões que estão em
`#include <stdlib.h>`

Declarar Matriz

```
float notas[4][3] = {  
    {5.0, 4.5, 7.0},  
    {9.6, 7.8, 0.0},  
    {2.5, 1.5, 9.5},  
    {10.0, 8.5, 7.5}  
};
```

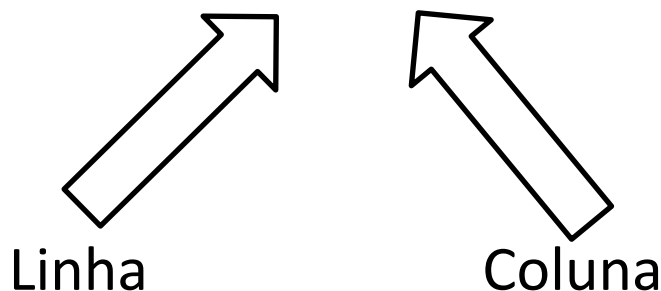


5.0	4.5	7.0
9.6	7.8	0.0
2.5	1.5	9.5
10.0	8.5	7.5

Matriz nada mais é do que um vetor de vetores

Acessar dados da Matriz

```
media = notas[0][0]*0.35 +  
        notas[0][1]*0.35 +  
        notas[0][2]*0.3
```



	[0]	[1]	[2]
[0]	5.0	4.5	7.0
[1]	9.6	7.8	0.0
[2]	2.5	1.5	9.5
[3]	10.0	8.5	7.5

Preencher Matriz

```
#include <stdio.h>

int main (){
    int i, j;
    float notas[4][3];
    for (i = 0 ; i < 4 ; i++) {
        for (j = 0 ; j < 3 ; j++) {
            printf("Nota %d do Aluno %d: ", (j+1),(i+1));
            scanf("%f", &notas[i][j]);
        }
    }
}
```

São necessários dois **for**, pois precisamos ver todas as linhas e colunas

Declarar Ponteiro

- Para declarar um ponteiro usamos *

```
int x, *y;
```

- x é um variável inteira
- y é um ponteiro para um inteiro

Acessar Ponteiro

- O operador & serve para acessar o endereço de uma variável e o operador * além da declaração serve para acessar o valor de um determinado endereço

```
int x, *y;  
y = &x;  
printf("%p e %p", &x, y);  
x = 10;  
printf("%d", *y);
```

- %p imprime um endereço de memória em hexadecimal

Aritmética de Ponteiros

- É possível realizar operações sobre os ponteiros, vale ressaltar que pode-se
 - Somar ou subtrair um inteiro de um ponteiro
 - Incrementar ou decrementar ponteiro
 - Subtrair ponteiros (produz um inteiro não um ponteiro)
 - Comparar ponteiros

```
int x[5] = {2,4,6,8,10};  
int *y;  
y = x;  
printf("%d\n", *(y+3));
```

Alocação Dinâmica

- Precisamos de funções do #include <stdlib.h>

Função	Utilidade
malloc()	Alocação inicial da memória
sizeof()	Tamanho de uma variável (em bytes) ou tipo de variável
free()	Libera espaço de memória de uma variável alocada

```
int *v, tam = 5;  
v = (int *)malloc(sizeof(int)*tam);  
free(v);
```

Indireção Múltipla (Matriz Dinâmica)

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

int main(){
    float **v;
    int i, j, n, m;
    scanf("%d", &n); // Linhas
    scanf("%d", &m); // Colunas
    v = (float **)malloc(n * sizeof(float *));
    for (i = 0 ; i < n ; i++)
        v[i] = (float *)malloc(m * sizeof(float));

    for (i = 0 ; i < n ; i++)
        for (j = 0 ; j < m ; j++)
            scanf("%f", &v[i][j]);

    return 0;
}
```

Definição de Funções

```
<tipo> <nome_da_função> (<parametros>) {  
    comando1  
    comando2  
    .  
    .  
    .  
    comandoN  
    return <variável com resultado>  
}
```

Exemplo Função

tipo da saída

nome da função

parâmetro 1

parâmetro 2

```
float func (float x, int n) {  
    float resultado = pow(x, (1.0/n));  
    return resultado;  
}
```

função em outra função
(pode sim)

variável com a saída da função
(mesmo tipo declarado no começo da função)

Passagem de Parâmetros

- Por valor

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

```
int func (int z) {  
    int y = 1;  
    z = 2;  
    return z*y;  
}
```

```
int main () {  
    int x = 1;  
    func(x);  
    printf("%d", x);  
}
```

- O valor do x será 1

- Por referência

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

```
int func (int *z) {  
    int y = 1;  
    *z = 2;  
    return (*z)*y;  
}
```

```
int main () {  
    int x = 1;  
    func(&x);  
    printf("%d", x);  
}
```

- O valor do x será 2

Definição de Struct

```
struct <nome> {  
    <tipo> <nome>;  
    <tipo> <nome>;  
    .  
    .  
    .  
    <tipo> <nome>;  
};
```

Acessar dados do Struct

```
struct usuario {  
    int id;  
    char login[50];  
    char senha[50];  
    char nome[100];  
};  
  
int main() {  
    struct usuario alunosCCT[1000];  
    for (i = 0 ; i < 1000 ; i++) {  
        scanf("%s", &alunosCCT[i].nome);  
    }  
}
```