

Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

SIN 110

Programação

Sistemas de Informação
Profa. Rachel Reis
rachel.reis@ufv.br

Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

Aula de Hoje

Algoritmos e Linguagem C

Créditos:

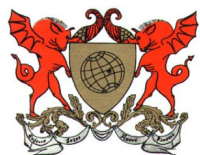
Prof. Guilherme Pena

Antes de escrever qualquer Algoritmo

Objetivo: Tenha o objetivo como foco. Um Algoritmo está correto se ele alcança seu objetivo.

Recursos: Conheça todos os recursos/informações que você já possui e pode utilizar durante o algoritmo.

Lógica: Faça um caminho mental dos passos a serem seguidos para se chegar ao objetivo utilizando os recursos disponíveis.



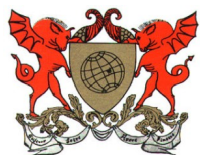
Linguagem de Programação C

A linguagem C foi criada por Dennis Ritchie (1941-2011), em 1972, no centro de Pesquisas da Bell Laboratories.

Sua primeira utilização importante foi a reescrita do Sistema Operacional UNIX, que até então era escrito na linguagem Assembly.

Características da linguagem C:

- Linguagem imperativa
- Simples e de fácil aprendizado



Normas Gerais: Caracteres Válidos

Um programa-fonte em C é um texto não formatado escrito em um editor de textos usando um conjunto padrão de caracteres ASCII.

Abaixo estão os caracteres utilizados em C:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z  
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
1 2 3 4 5 6 7 8 9 0  
+ - * / \ = | & ! ? # % ( ) { } [ ] _ ' " . , ; : < >
```



Algoritmos Computacionais

Quando falamos de algoritmos computacionais, ou algoritmos descritos para programas, existem alguns conceitos pertinentes:

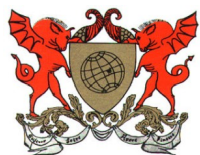
- Comandos
- Variáveis
- Tipo de dados
- Identificadores
- Operadores
- Estruturas
- etc

Comandos

Por definição, um comando representa alguma ação que o algoritmo fará.

Em geral, e como boa prática de descrição, cada comando que representa ações diferentes, ocupa uma linha distinta na descrição.

Isso facilita a compreensão do algoritmo, legibilidade.



Comandos

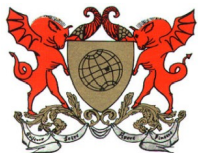
Por exemplo:

```
#include <stdio.h>
int main(){
    int n1, n2, m;
    printf("Digite dois números:");
    scanf("%d", &n1);
    scanf("%d", &n2);
    m = n1 * n2;
    printf("Multiplicação = %d", m);
    return 0;
}
```

Normas Gerais: Ponto e Vírgula

Todos comandos (com exceção de algumas estruturas) que forem escritos na linguagem C devem terminar com o caracter:

;



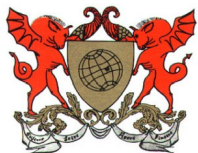
A Função **main()**

A função **main()** é o ponto de **início da execução** do programa e deve existir em algum lugar dentro do seu programa.

Exemplo:

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

int main()          // definição do programa principal
{
    printf("Hello, World!"); // escreve a mensagem na tela
    return 0;          // finaliza o programa principal
}
```



A Função **main()**

Quando a função **main()** termina, o programa deve passar uma instrução para o sistema operacional informando o fato.

Essa instrução é o comando:

```
return 0;
```

No Dev-C++, o ***return 0;*** fecha a execução do programa, logo, devemos usar um comando de pausa antes do ***return 0;*** para impedir isso:

```
system("pause");
```

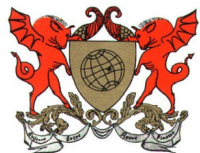


Variável

Um algoritmo, e posteriormente um programa, recebe **dados**.

Os dados precisam ser armazenados no computador, para que possam ser utilizados no processamento.

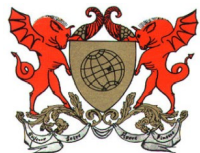
O armazenamento dos dados é feito na memória.



Variável

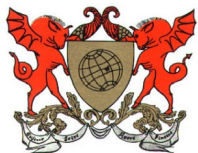
Os dados são armazenados em locais específicos da memória, esses locais possuem endereços denominados **ENDEREÇOS DE MEMÓRIA**.

Por exemplo: quando uma operação aritmética recebe dois operandos, cada operando é armazenado em um endereço de memória **diferente**, para ser utilizado no cálculo.



Variável

A memória do computador atua como um grande guarda-volumes etiquetado.



Variável

Como cada espaço de memória pode armazenar dados várias vezes, ou seja, seu conteúdo pode variar, chamamos estes espaços de memória de variáveis.

Cada **variável** representa uma **posição/endereço** de memória, e possui um **nome** e um **tipo** (tipo de dado que ela armazena).

O conteúdo da variável pode variar ao longo do tempo, durante a execução de um programa. No entanto, **o tipo não varia**.



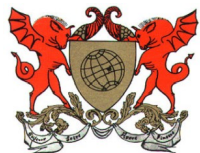
Tipos de Dados

Os tipos de dados determinam a qual classe representativa um determinado dado irá pertencer.

Em outras palavras, um tipo define quantos **bytes de memória** uma determinada **variável** ocupará.

Os tipos de dados principais são:

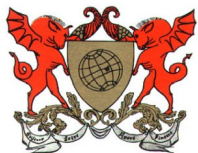
- Numérico (valor inteiro ou real)
- Lógico
- Literal ou caractere



Tipos de Dados

Numérico (inteiro):

- Podem representar números inteiros
- Os números **inteiros** podem ser negativos ou positivos, e ocupam 1, 2 ou 4 bytes de memória no computador.
- Isso permite armazenar números entre:
 - 1 byte : -256 e +255
 - 2 bytes: -32.767 e +32.768
 - 4 bytes: -2.147.483.648 e +2.147.483.647



Tipos de Dados

Numérico (real):

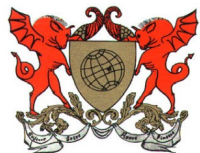
- Podem representar números inteiros ou reais
- Os números **reais** podem ser negativos ou positivos, e ocupam 4 ou 8 bytes de memória no computador.
- Isso permite armazenar números com diferentes tamanhos e precisões de casas decimais.



Tipos de Dados

Lógico:

- Também chamados de *booleanos*
- Podem assumir os valores VERDADEIRO e FALSO
- Em geral, ocupam apenas 1 byte de memória



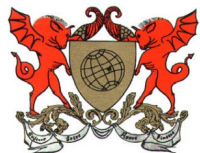
Tipos de Dados

O **tipo lógico** não existe em C.

No entanto, C possui um mecanismo que assume o seguinte:

- 0 = **FALSO**
- qualquer outro valor = **VERDADEIRO**

Logo, usamos o tipo **inteiro** para representar o tipo lógico usando valores **0** para **FALSO** e **(diferente de 0)** para **VERDADEIRO**.

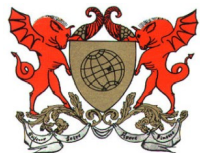


Tipos de Dados

Caractere ou *caracter*:

- Dados formados por um caractere, ou uma cadeia de caracteres
- Ocupam apenas 1 byte de memória **por caractere**

a b c d e ... A B C D E ... 1 2 3 4 5 ... ! @ # \$
% ...



Tipos de Dados em Linguagem C

O **tipo** de uma variável informa a quantidade de memória, em bytes, que esta irá ocupar e a forma como seu conteúdo será armazenado.

Tipos básicos em C:

Tipo	Bit	Bytes
char	8	1
int	32	4
float	32	4
double	64	8



Tipos de Dados em C

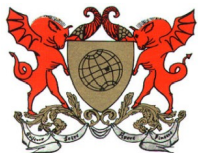
Valores:

- Quando estamos trabalhando com qualquer valor puro na descrição do algoritmo, damos a este o nome de “**CONSTANTE**”, seja de qualquer tipo.

```
X = 10;
```

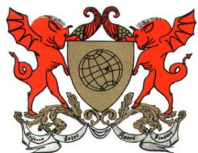
```
letra = 'A';
```

```
pi = 3.14;
```



Identificadores

- Representam os nomes das variáveis, dos programas, etc.
- Regras básicas para formação de nomes:
 - Caracteres permitidos: números, letras (maiúsculas ou minúsculas) e o caractere sublinhado “_”.
 - O primeiro caractere dever ser sempre **uma letra ou o caractere sublinhado**.
 - Não são permitidos caracteres em branco, caracteres especiais (@, \$,+,-,%,!) ou **caracteres com acentuação!**
 - Não é possível utilizar palavras reservadas nos identificadores, ou seja, palavras que pertencem à **linguagem de programação** utilizada.

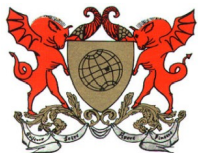


Palavras Reservadas

São utilizadas pela linguagem C e não podem jamais serem usadas como identificadores.

Lista de Palavras Reservadas (32)

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



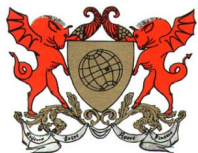
Identificadores

Exemplos permitidos:

A, a, Nota, nota, nota_1, dia, _soma

Exemplos não-permitidos:

1A, Á, à, Not@, not-a, 1nota_1, #hashtag

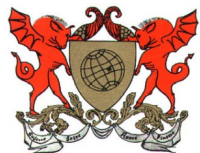


Declaração de Variáveis

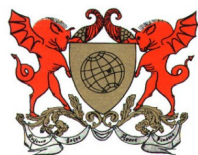
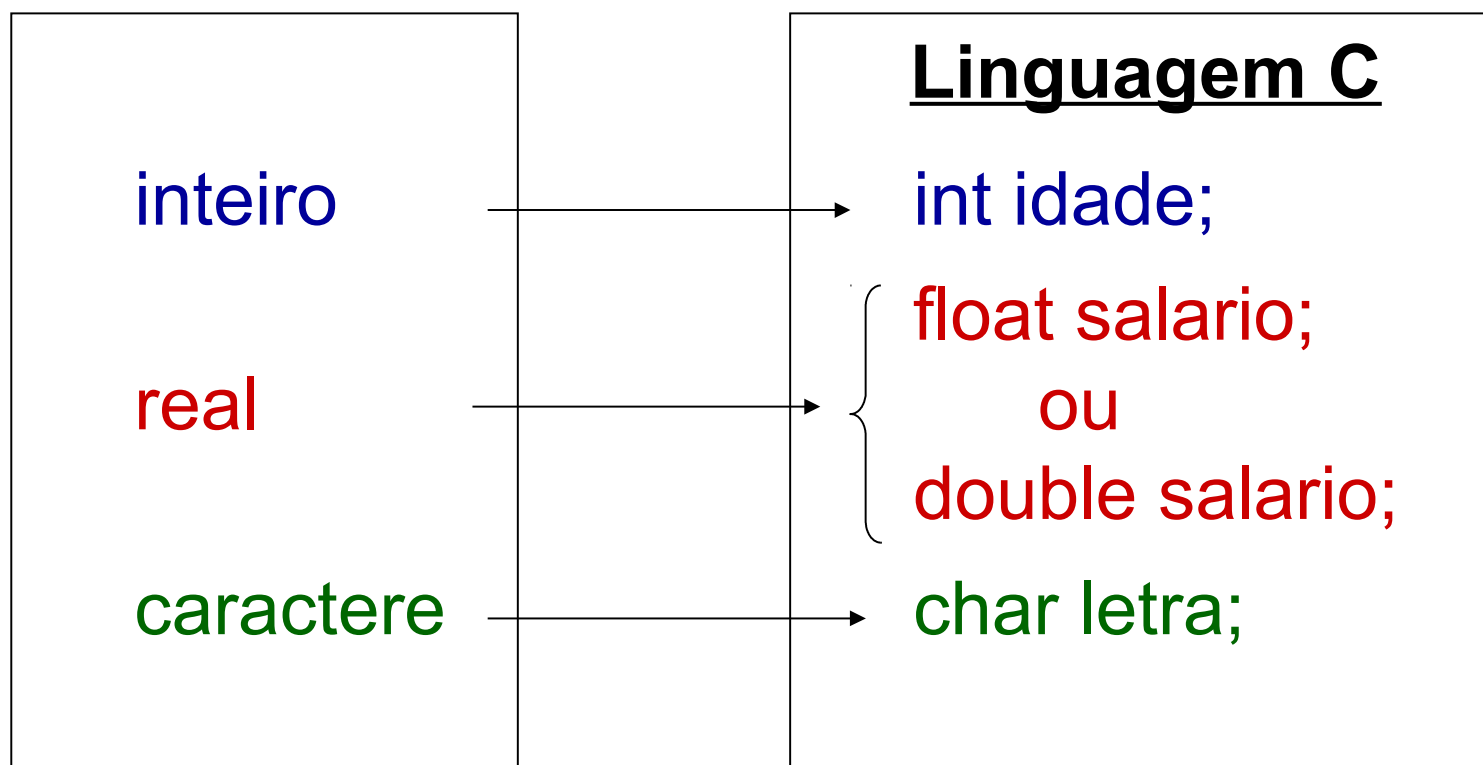
Uma declaração de variável é uma instrução para reservar uma quantidade de memória para armazenar um tipo especificado de dado.

A declaração de uma variável consiste de um **tipo** e um **identificador**

- O **tipo** determina o espaço de memória que deverá ser alocado (tipo do valor que ela armazena, ex.: int, char)
- O **identificador** permitirá que ela seja referenciada no restante do programa (nome/apelido).



Declaração de Variáveis em C

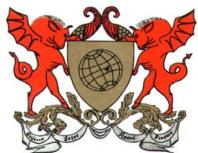


Declaração de Variáveis em C

Várias variáveis são declaradas na mesma linha separando os nomes por vírgula:

,

```
int main( )  
{  
    int i, j, k, numero;  
    float x, y, z;  
    char letra, l1, l2, l3;  
    double r1, r2, r3;  
}
```



Atribuição

É a **principal forma de se armazenar um dado** em uma variável.

Esse comando permite que você forneça um valor a uma variável, onde o **tipo desse valor tem que ser compatível com a variável**.

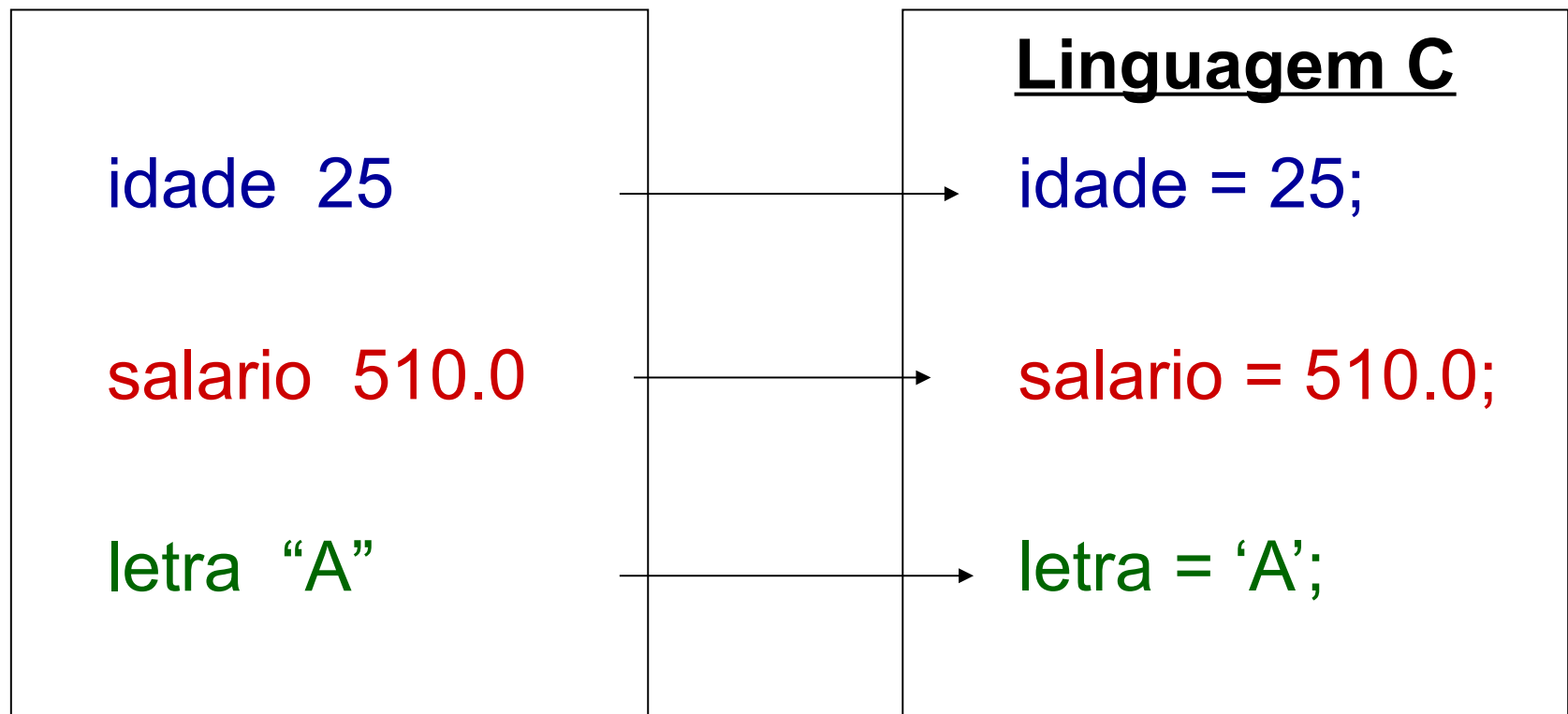
O comando de atribuição é representado por **= (sinal de igualdade)**

Exemplo: **x = 10;**

Lê-se: “A variável x recebe o valor 10 ou x recebe o valor 10”



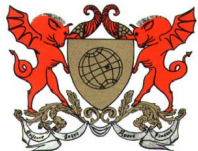
Atribuição



Inicialização de variáveis

Em C, no ato de declarar uma variável também é possível inicializar o seu valor com o comando de atribuição.

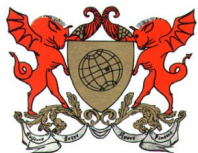
```
int main( )  
{  
    int i = 0, j = 0, numero;  
    char letra1 = 'a', letra2 = 'b';  
    float x = 1.1, n_real;  
}
```



Comentários

Comentários podem ser escritos em qualquer lugar do código.

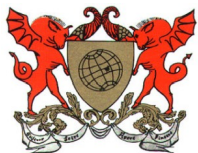
```
int main( )  
{  
    /* comentário de  
       várias linhas */  
  
    // comentário de uma linha  
}
```



Comentários

Comentários são úteis para documentação e explicação de código.

```
#include <stdio.h>
int main()
{
    // Declaração da variável nota
    float nota;
    /* A instrução abaixo atribui o valor 7.5 à
       variável nota */
    nota = 7.5;
    return 0;
}
```



Estrutura Sequencial

- Os comandos de um algoritmo fazem parte de uma **seqüência**, onde é **relevante a ordem** na qual se encontram os mesmos.

Pois os comandos serão executados um de cada vez, **estritamente**, de acordo com essa ordem.



Estrutura Sequencial (C)

Bibliotecas

int main()

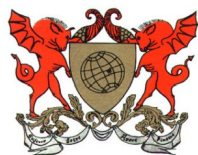
Comando - 1

Comando - 2

:

Comando - N

}



Bibliotecas em C

Bibliotecas, são coleções de comandos pré-definidos que são muito úteis no desenvolvimento das aplicações em uma linguagem de programação.

Para usar uma **biblioteca**, usa-se:

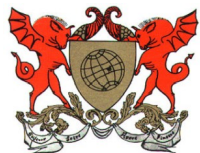
Exemplos: `#include < NOME_DA_BIBLIOTECA >`

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

Saída de dados

Saída

- O comando de saída é utilizado para mostrar dados para o usuário, em geral, na tela do monitor.
- Pode-se usar qualquer outro dispositivo de saída caso a linguagem dê suporte.



Saída - Função printf()

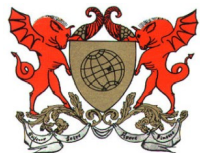
Responsável pela impressão de dados na tela do computador.

Sintaxe:

```
printf("Expr. de controle", lista de argumentos);
```

Para usar essa função é necessário incluir o que chamamos de **biblioteca**, que inclui ferramentas úteis ao programa:

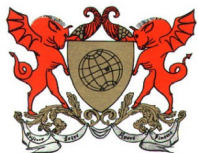
```
#include <stdio.h>
```



Função printf()

Exemplo:

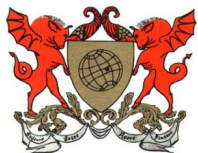
```
#include <stdio.h>
int main( )
{
    printf("Meu primeiro programa em C");
    return 0;
}
```



Função printf()

Código para **impressão formatada**.

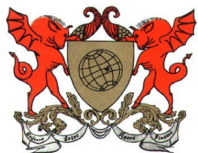
<code>%c</code>	um único caracter.
<code>%o, %d, %x</code>	um número inteiro em octal, decimal ou Hexadecimal.
<code>%u</code>	um número inteiro em base decimal sem sinal.
<code>%ld</code>	um número inteiro longo em base decimal.
<code>%f, %lf</code>	um número real de precisão simples ou dupla.
<code>%s</code>	uma cadeia de caracteres (string).
<code>%%</code>	um único sinal de porcentagem.



Função printf()

Caracteres de controle utilizados com a função printf()

<code>\a</code>	soa o alarme do microcomputador (beep)
<code>\b</code>	o cursor retrocede uma coluna.
<code>\f</code>	Alimentação de formulário (FF).
<code>\n</code>	o cursor avança para uma nova linha.
<code>\r</code>	o cursor retrocede para o início da linha.
<code>\t</code>	o cursor avança para próxima marca de tabulação.
<code>\"</code>	exibe aspas duplas.
<code>\'</code>	exibe aspas simples.
<code>\\</code>	exibe uma única barra invertida.



Função printf()

Exemplo:

```
#include <stdio.h>
int main( )
{
    printf("Este e o numero dois: %d\n", 2) ;
    return 0;
}
```

O programa imprimirá na tela:

```
Este e o numero dois: 2
```

Função printf()

Exemplo:

```
#include <stdio.h>
int main( )
{
    printf("Linha %d\nLinha %d\nLinha %d\n", 1, 2, 3);
    return 0;
}
```

O programa imprimirá na tela:

```
Linha 1
Linha 2
Linha 3
```



Função printf()

Exemplo:

```
#include <stdio.h>
int main( )
{
    printf("Dia %d de %s e feriado!\n", 7, "setembro");
    return 0;
}
```

O programa imprimirá na tela:

Dia 7 de setembro e feriado!



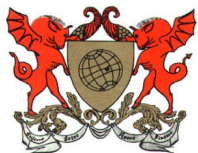
Função printf()

Exemplo: Formantando o número de casas decimais do tipo float.

```
#include <stdio.h>
int main( )
{
    float nota = 7.5;
    printf("A nota do aluno é: %f\n",nota) ;
    printf("A nota do aluno é: %0.2f\n",nota) ;
    return 0;
}
```

O programa imprimirá na tela:

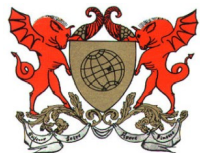
```
A nota do aluno é: 7.500000
A nota do aluno é: 7.50
```



Entrada de dados

Entrada

- O comando de entrada é utilizado para receber dados informados pelo usuário, em geral, através do teclado.
- Pode-se usar qualquer outro dispositivo de entrada caso a linguagem dê suporte.



Entrada - Função scanf()

Responsável pela leitura de dados através do teclado.

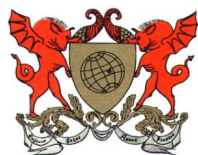
Sintaxe:

```
scanf("Expr. de controle", lista de argumentos);
```

Para usar essa função também é necessário incluir a **biblioteca**:

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

Na expressão de controle, **indicamos o tipo de cada variável** sendo lida na ordem. (Tipos: %d, %c, %f ...)



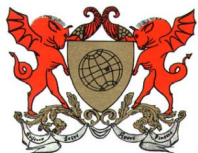
Função scanf()

IMPORTANTE:

Quando digitamos um valor, estamos lendo valores do teclado para variáveis declaradas previamente.

Em C, para isso devemos usar o operador de endereço: **&**

que indica a posição da variável na memória, logo o valor digitado é armazenado no local correto.



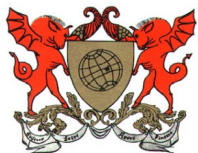
Função scanf()

Exemplo: Leitura de um valor inteiro.

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

O programa imprimirá na tela:

```
Digite um numero inteiro: 56
O valor digitado foi: 56
```



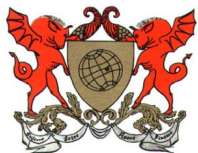
Função scanf()

Exemplo: Leitura de apenas uma letra.

```
#include <stdio.h>
int main( )
{
    char letra;
    printf("Digite uma letra do alfabeto: ");
    scanf("%c", &letra);
    printf("A letra digitada foi: %c\n", letra);
    return 0;
}
```

O programa imprimirá na tela:

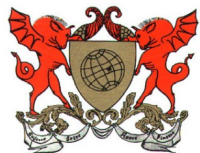
```
Digite uma letra do alfabeto: H
A letra digitada foi: H
```



Função scanf() - fflush(stdin)

A função scanf() tem um detalhe importante na linguagem C ao ler **char** ou **strings**:

Sempre que digitamos um valor, apertamos a tecla **ENTER**, e esse **ENTER** fica gravado em um espaço, chamado *buffer* de entrada, como um caracter '**\n**'.

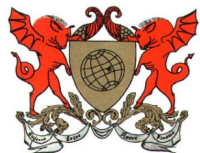


Função scanf() - fflush(stdin)

O grande problema desse caracter '\n' é que no primeiro uso de um scanf() (seja ler um **inteiro**, um **real** ou um **char**) e depois precisamos ler novamente outro dado char:

```
...  
scanf ("%d", &numero) ;  
scanf ("%c", &letra) ;  
...
```

O segundo scanf() percebe o '\n' deixado pelo primeiro scanf() como um caracter válido e “**pega**” ele para a variável do segundo scanf().



Função scanf() - **fflush(stdin)**

A forma de solucionar este problema é usando uma função especial **fflush(stdin)** que “limpa” o *buffer* de entrada, tirando o caracter '\n' de lá.

```
...  
scanf ("%d", &numero) ;  
fflush(stdin) ;  
scanf ("%c", &letra) ;  
...
```

Nesse caso, com o *buffer* limpo, o segundo scanf() volta a ficar esperando que o usuário digite alguma coisa.



Função scanf() - **fflush(stdin)**

OBS: Lembre-se que isso só será necessário se o programa precisar ler um **char** ou uma **string** após um scanf() já ter sido acionado.

Se o programa ler **apenas** valores numéricos (int ou float) então não precisa usar a função especial.

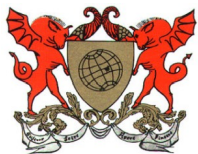


Função scanf()

Uma outra forma de solucionar o mesmo problema é usar um caracter '**espaço**' antes do **%c**.

```
...  
scanf ("%d", &numero);  
scanf (" %c", &letra);  
...
```

Nesse caso, o **buffer não é limpo**, no entanto o caracter '**espaço**' consome o pulo de linha que estiver no buffer e o programa volta a ficar esperando que o usuário digite alguma coisa.



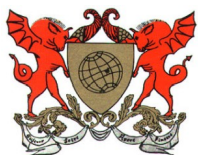
Função scanf() - fflush(stdin)

Exemplo:

```
#include <stdio.h>
int main( )
{
    int num;
    char letra;
    printf("Digite um numero inteiro: ");
    scanf("%d", &num);
    printf("O valor digitado foi: %d\n", num);
    printf("Digite uma letra do alfabeto: ");
    fflush(stdin);
    scanf("%c", &letra);
    printf("A letra digitada foi: %c\n", letra);
    return 0;
}
```

O programa imprimirá na tela:

```
Digite um numero inteiro: 56
O valor digitado foi: 56
Digite uma letra do alfabeto: H
A letra digitada foi: H
```



Função scanf()

Exemplo:

```
#include <stdio.h>

int main( )
{
    int num;
    char letra;
    printf("Digite um numero inteiro: ");
    scanf("%d", &num);
    printf("O valor digitado foi: %d\n", num);
    printf("Digite uma letra do alfabeto: ");
    scanf(" %c", &letra);
    printf("A letra digitada foi: %c\n", letra);
    return 0;
}
```

O programa imprimirá na tela:

```
Digite um numero inteiro: 56
O valor digitado foi: 56
Digite uma letra do alfabeto: H
A letra digitada foi: H
```



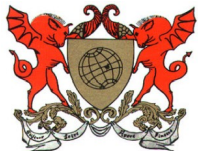
Função scanf() - CUIDADO!

Apesar da semelhança com printf(), na função scanf() **NÃO** se usa **“textos”** dentro dos parênteses, apenas os tipos **“%”** para os valores que serão digitados:

```
#include <stdio.h>

int main( )
{
    int num;
    scanf("Digite um numero inteiro: %d", &num);
    printf("O valor digitado foi: %d\n", num);
    return 0;
}
```

```
printf("Digite um numero inteiro: ");
scanf("%d", &num);
```



Exemplos: printf() e scanf()

Exemplo:

```
#include <stdio.h>
int main( )
{
    int anoNasc;

    // Texto: "Digite o ano do seu nascimento: "

    // Leia o valor digitado

    // Imprima o ano de nascimento

    return 0;
}
```



Exemplos: printf() e scanf()

Exemplo:

```
#include <stdio.h>
int main( )
{
    int anoNasc;

    printf("Digite o ano do seu nascimento: ");

    scanf("%d", &anoNasc);

    printf("\nAno de Nascimento: %d", anoNasc);

    return 0;
}
```



Exemplos: printf() e scanf()

Exemplo:

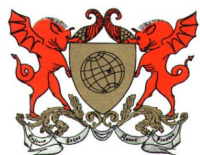
```
#include <stdio.h>
int main( )
{
    char conceito;

    // Texto: "Qual seu conceito final em Programação I?"

    // Leia o valor digitado

    // Imprima o conceito

    return 0;
}
```



Exemplos: printf() e scanf()

Exemplo:

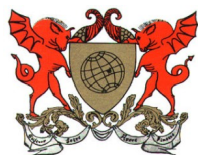
```
#include <stdio.h>
int main( )
{
    char conceito;

    printf("Qual seu conceito final em Programação I?");

    scanf("%c", &conceito);

    printf("\nConceito: %c", conceito);

    return 0;
}
```



Exemplos: printf() e scanf()

Exemplo:

```
#include <stdio.h>
int main( )
{
    int num1, num2;

    // Texto: "Digite dois numeros"

    // Leia os valores digitados

    // Imprima os dois numeros

    return 0;
}
```



Exemplos: printf() e scanf()

Exemplo:

```
#include <stdio.h>
int main( )
{
    int num1, num2;

    printf("Digite dois números: ");

    scanf("%d", &num1);
    scanf("%d", &num2);

    printf("Números: %d, %d", num1, num2);

    return 0;
}
```



Exemplos: printf() e scanf()

Exemplo:

```
#include <stdio.h>
int main( )
{
    int num1, num2;

    printf("Digite dois números: ");

    scanf("%d %d", &num1, &num2);

    printf("Números: %d, %d", num1, num2);

    return 0;
}
```

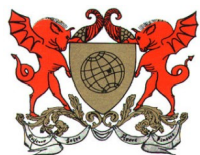


Operadores

Operadores referem-se a alguns símbolos que podem ser usados na descrição do algoritmo.

Os operadores também pode ser divididos em classes:

- Operadores aritméticos
- Operadores relacionais
- Operadores lógicos
- Operadores gerais (atribuição, declaração, etc..)
- entre outros



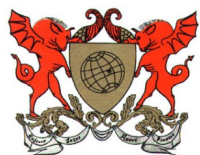
Operadores Aritméticos em C

Operadores Binários:

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

Operador Unário:

-	Menos unário (troca de sinal do valor)
---	--



Operadores Aritméticos em C

Operador de Atribuição =

- Representa a atribuição **da expressão a direita ao nome da variável a esquerda**. Exemplo: num = 2000;

Operador + - / *

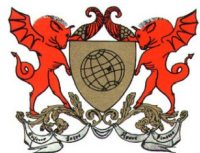
- Representam as operações aritméticas básicas de soma, subtração, divisão e multiplicação.

Operador menos unário –

- Usado somente para indicar a troca do sinal algébrico do valor.

Operador módulo %

- Retorna o resto da divisão inteira.



Precedência de Operadores Aritméticos

A **precedência de operadores** é importante para que a expressão aritmética retorne o resultado correto.

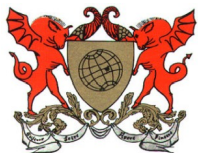
Além disso, **os parênteses “ () ” mais internos** nas expressões tem precedência sobre outros parênteses e sobre os operadores.

Linearização de Expressões

Na construção de algoritmos que realizam cálculos matemáticos, **todas** as expressões aritméticas devem ser **linearizadas**, ou seja, colocadas na mesma linha:

Exemplo:

$\left\{ \left[\frac{2}{3} - (5 - 3) \right] + 1 \right\} . 5$	$((2/3 - (5 - 3)) + 1) * 5$
Tradicional	Computacional



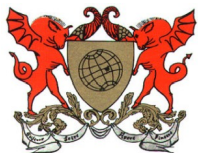
Operadores Aritméticos em C

Exemplo:

“Ler uma temperatura em graus Fahrenheit e apresentá-la convertida em graus Celsius . A fórmula de conversão é: $C = (F - 32.0) * (5.0/9.0)$, sendo F a temperatura em Fahrenheit e C a temperatura em Celsius”

Pseudocódigo:

```
var
    F, C : real
inicio
    escreva("Digite a temperatura em Fahrenheit: ")
    leia(F)
    C ← (F - 32.0) * (5.0/9.0)
    escreva("A temperatura em graus celsius é: ", C)
fimAlgoritmo
```

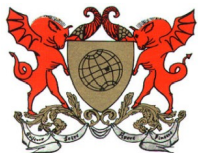


Operadores Aritméticos em C

Programa em C:

```
#include <stdio.h>

int main( )
{
    float F, C;
    printf("Digite a temperatura em Fahrenheit: ");
    scanf("%f", &F);
    C = (F - 32.0) * (5.0/9.0);
    printf("A temperatura em graus celsius e: %f\n", C);
    return 0;
}
```



Operadores e função printf()

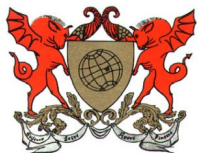
A função printf() também aceita operações na lista de argumentos e o resultado mostrado na tela será do tipo que foi especificado para a saída formatada.

```
#include <stdio.h>

int main( ){
    float n1 = 5.5, n2 = 2.0;
    printf("A nota do aluno é: %f\n", n1 + n2);
    printf("A nota do aluno é: %0.2f\n", n1 + n2);
    return 0;
}
```

O programa imprimirá na tela:

```
A nota do aluno é: 7.500000
A nota do aluno é: 7.50
```

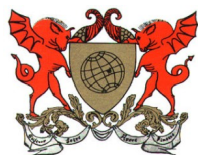


Teste de Mesa (Rastreio)

Depois que elaboramos um programa podemos (e devemos) testá-lo. Um método conhecido é o **Teste de Mesa (ou rastreio)** do programa.



Teste de Mesa (Rastreio)



Teste de Mesa (Rastreio)

Basicamente, **cria-se uma tabela** onde, para cada **variável** você deve fazer **uma coluna** e **uma coluna para saída de dados**.

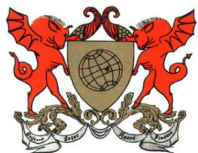


Teste de Mesa (Rastreio)

As linhas da tabela vão sendo preenchidas à medida que os comandos vão sendo executados e, os valores das colunas vão sendo atualizados.

Exemplo: Faça o teste de mesa do programa ao lado.

```
int main(){
    int A, B, C;
    A = 5;
    B = 15;
    C = A + B;
    printf("%d\n", C);
    A = 10;
    B = 25;
    C = A + B;
    printf("%d\n", C);
    A = A - B;
    printf("%d\n", A);
    A = 0;
    B = 0;
    C = 0;
    return 0;
}
```



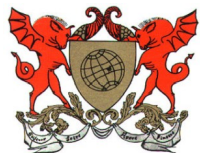
Exercícios

- 1) Faça um programa que leia dois números inteiros e imprima na tela a soma dos dois números lidos.
- 2) Faça um programa que leia dois números reais e imprima na tela a multiplicação dos dois.
- 3) Faça um programa em que o usuário digite 3 letras e imprima as 3 letras todas em uma mesma linha.



Exercícios

- 4) Escreva um algoritmo que receba três notas e seus respectivos pesos, calcule e mostre a média ponderada entre essas notas.
- 5) Escreva um algoritmo que receba o salário de um funcionário, calcule e mostre o novo salário, sabendo-se que este sofreu um aumento de 25%.



Exercícios

- 6) Leia um valor em real e a cotação do dólar. Em seguida, imprima o valor correspondente em dólares.
- 7) Leia um número inteiro e imprima a soma do sucessor de seu triplo com o antecessor de seu dobro.
- 8) Leia um número inteiro de 4 dígitos e imprima 1 dígito por linha.

