

Introdução à Linguagem C

Prof^a.Dr^a.Thatyana de Faria Piola Seraphim (ECO)
Prof.Dr.Enzo Seraphim (ECO)

Universidade Federal de Itajubá

thatyana@unifei.edu.br
seraphim@unifei.edu.br

Linguagem C

Introdução - História

- ▶ A linguagem de programação C foi projetada para permitir grande economia de expressão nos programas, isto é, produzir programas fonte mais compactos.
- ▶ Foi usada para escrever cerca de 90% do código do sistema operacional UNIX
 - ▶ com a popularização do UNIX em equipamentos de médio porte, e até micros, a linguagem C também ganhou popularidade entre os programadores profissionais.
- ▶ **1969**: os laboratórios Bell lançaram uma versão básica do sistema operacional UNIX escrito em *Assembly*
 - ▶ *Keneth Thompson* desenvolveu em 1969 uma linguagem experimental chamada **B**
- ▶ **1972**: a partir da linguagem **B**, a linguagem C foi projetada

Linguagem C

Introdução - Criadores do Unix e Linguagem C (1969)



Ken Thompson (1943)



Dennis Ritchie (1941-2011)

Linguagem C

Introdução - História

- ▶ **1973**: o sistema operacional UNIX foi melhorado e cerca de 90% de seu código foi escrito em C.
- ▶ Por causa da libertação do *Assembly*, o UNIX (e consequentemente C) adquiriu grande portabilidade:
 - ▶ foi rapidamente adaptado a uma série de computadores e seu uso não parou de crescer
- ▶ No final da década de 70 e início da década de 80
 - ▶ a proliferação de UNIX e C foi muito grande e chegou até os micros.
 - ▶ C ficou independente do sistema operacional UNIX e uma série de compiladores C surgiram para muitos equipamentos

Linguagem C

Introdução - História

- ▶ C é a linguagem preferida dos programadores profissionais por várias razões
 - ▶ chega a substituir *Assembly* em boa parte do *software* recentemente desenvolvido;
 - ▶ para muitos programadores que há alguns anos, vinham desenvolvendo seus programas em *Assembly*, estão hoje quase todos escritos em C.
- ▶ Com o advento da Programação Orientada para Objetos (*OOP-Object Oriented Programming*)
 - ▶ a linguagem que se tornou mais usada para esta técnica de programação é uma extensão da linguagem C, chamada C++.
- ▶ A maneira de comunicação com um computador chama-se **programa** e a única linguagem que o computador entende chama-se **linguagem de máquina.**

Linguagem C

Introdução - Criadores do Linux e GCC



Linus Torvalds
(Linux – 1991)



Richard Stallman
(Projeto Gnome – 1983)

Linguagem C

Introdução - Organização dos Programas em C

C

Assembly

Binário

<code>int x = 10;</code>	<code>ldaa 10 // carrega 10 no acum. a</code>	<code>0x83 0x0a</code>
<code>int y = 20;</code>	<code>staa x // salva o valor do acum. a em x</code>	<code>0x84 0x00</code>
<code>int z = x*y;</code>	<code>ldaa 20 // carrega o valor 20 no acum. a</code>	<code>0x83 0x14</code>
	<code>staa y // salva o valor do acum. a em y</code>	<code>0x84 0x01</code>
	<code>ldaa x // carrega o acum. a com o valor de x</code>	<code>0x83 0x00</code>
	<code>ldab y // carrega o acum. b com o valor de y</code>	<code>0x93 0x01</code>
	<code>mulab // mult. acum. a por b, salva em a</code>	<code>0x5f</code>
	<code>staa z // salva o valor do acum. a em z</code>	<code>0x83 0x02</code>

Linguagem C

Introdução - Alice no mundo das maravilhas

Mundo Real: Devastação ocasionada pelos tornados

Mundo Maravilhoso: Coelho, Chapeleiro, Lagarta e a Rainha de Copas



Mundo Real x Mundo Maravilhoso

Linguagem Assembly x Linguagem C

Linguagem C

Introdução - Organização dos Programas em C

- ▶ Um programa C é uma coleção de funções criadas pelo programador ou funções de biblioteca.
- ▶ A grande maioria dos compiladores vem com uma grande quantidade de funções já criadas e compiladas em biblioteca que são usadas dependendo da necessidade do programador.
- ▶ Os componentes de um programa em C são:
 - ▶ **Comentários:** podem e devem estar em qualquer ponto do programa. São escritos entre os delimitadores `/*` e `*/` para um bloco de comandos ou utilizar `//` para comentar uma linha

Comentários

```
1  /* Programa 01 */  
2  /* Funcao: descricao */  
3  // Autor: nome
```

Linguagem C

Introdução - Organização dos Programas em C

- ▶ Diretivas de compilação: não são instruções próprias da linguagem C. São mensagens que o programador envia ao compilador para que este execute alguma tarefa no momento da compilação.
 - ▶ As diretivas são iniciadas pelo caractere #.
 - ▶ As diretivas mais comuns são #include e #define, ambas utilizadas para especificar bibliotecas de funções a serem incorporadas na compilação.

Diretivas

```
1 //Diretivas de compilacao
2 #include<biblioteca.h>
3 #define macros
4 #define labels
```

- ▶ **Definições Globais**: são especificações de constantes, tipos e variáveis que serão válidas em todas as funções que formam o programa.
 - ▶ Embora sejam de relativa utilidade, não é uma boa prática de programação definir muitas variáveis globais.
 - ▶ Podem ser acessadas em qualquer parte do programa.
 - ▶ Um breve descuido na alteração dos seus valores, pode provocar problemas em muitos outros locais.

Definições Globais

```
1 // Secao de variaveis globais
2 char variavelGlobal;
```

- ▶ **Protótipos de funções**: não são obrigatórios. São usados pelo compilador para fazer verificações durante a compilação.
 - ▶ Ver se as partes do programa que acionam as funções o fazem de modo correto, com o nome certo, com o número e tipo de parâmetros adequados.

Protótipo de Funções

```
1 // Secao de prototipo de funcoes
2 void funcao01(char var);
3 int funcao02(void);
```

- ▶ **Definições de funções:** são os blocos do programa onde são definidos os comandos a serem executados em cada função.
 - ▶ A função pode, ou não, receber valores que serão manipulados em seu interior.
 - ▶ Após o processamento, as funções podem, se necessário retornar um valor.
 - ▶ É obrigatório a presença de pelo menos uma função com o nome **main**, e esta será a função por onde começa a execução do programa.
 - ▶ Não há ordem obrigatória para codificar as funções.

Linguagem C

Introdução - Organização dos Programas em C

Definição de Funções

```
1 // Secao de definicao de funcoes
2 int main(int argc, char *argv[]) {
3     return 0;
4 } //end main
5 void funcao01(char var) {
6     ...
7 } //end funcao01
8 void funcao02(void) {
9     ...
10 } //end funcao02
```

main()

→ inicio de um bloco de instruções

int main() {
 ...
 return 0;
}

Linguagem C

Diretiva **include**

- ▶ **include**: serve para especificar ao compilador que deseja-se usar novas funções, tipos e macros que estão disponíveis em outros arquivos.
- ▶ Como a linguagem C tem uma grande variedade destas funções e definições
 - ▶ é comum que elas sejam agrupadas em arquivos diferentes, de acordo com a natureza das tarefas que elas executam.

Sintaxe

```
1 #include <nome do arquivo> .h
2 #include "nome do arquivo"
```

Linguagem C

Diretiva **include**

- ▶ A diferença entre as duas formas está no local onde o compilador vai procurar o arquivo no momento da compilação.
- ▶ **#include**<nome do arquivo>: o arquivo é procurado no diretório definido pelo compilador C, como sendo aquele que contém os *header files*.
 - ▶ *Header files*: são os arquivos com extensão .h que contêm as definições de tipos, dados e várias funções já prontas.
- ▶ **#include** "nome do arquivo": é usado quando deseja que o compilador busque o arquivo especificado no mesmo diretório do arquivo que está sendo criado.
 - ▶ Esta forma é usada quando deseja-se incorporar arquivos criados e salvos pelo programador no mesmo diretório atual.

Alguns arquivos (bibliotecas) padronizadas em todos os compiladores, compatíveis com a norma ANSI-C.

- ▶ **assert.h**: define a macro *assert()* que implementa uma asserção, utilizada para verificar suposições feitas pelo programa.
- ▶ **ctype.h**: rotinas para verificação de tipos de dados.
- ▶ **errno.h**: fornece macros para identificar e relatar erros de execução através de códigos de erro. A macro *errno* fornece um número inteiro positivo contendo o último código de erro fornecido por alguma função ou biblioteca que faz uso do *errno*.
- ▶ **float.h**: define macros que especificam características do ponto flutuante.
- ▶ **limits.h**: define macros que definem os limites dos tipos de dados.

Linguagem C

Diretiva **include**

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

- ▶ **math.h**: funções matemáticas.
- ▶ **stdio.h**: rotinas de entrada e saída definidas pelos criadores da linguagem C.
- ▶ **stddef.h**: vários tipos de dados e macro substituições.
- ▶ **stdlib.h**: possui funções envolvendo alocação de memória, controle de processos, conversões e outras.
- ▶ **string.h**: fornece funções, macros e definições da biblioteca padrão da linguagem de programação C para manipulação de cadeias de caracteres e regiões de memória.
- ▶ **time.h**: fornece protótipos para funções, macros e definição de tipos da biblioteca padrão da linguagem de programação C para manipulação de datas e horários de modo padrão.

Linguagem C

Variáveis



Uma variável em linguagem C possui:

- ▶ Um tipo que indica o tamanho.
- ▶ Um nome para referenciar o conteúdo.
- ▶ Um espaço reservado na memória para armazenar seu valor.

Variável

É um espaço de memória contém um valor o qual pode ser alterado ao longo do tempo.

Linguagem C

Variáveis - Identificadores ou nomes

- ▶ Para nomes de variáveis podem ser usados quantos caracteres forem desejados contanto que o primeiro caracter seja uma letra ou sublinhado.
- ▶ A linguagem C faz distinção entre maiúsculas e minúsculas. Por exemplo: matrix e MaTrIx são variáveis distintas.
- ▶ É comum utilizar apenas minúsculas para nomes de variáveis e apenas MAIÚSCULAS para constantes.
- ▶ Uma variável não pode ter o mesmo nome de uma palavra chave em linguagem C .

Linguagem C

Variáveis - Tipos de Dados e Tamanhos

Existem quatro tipos básicos em C:

- ▶ char - 1 byte: de -128 à 127;
- ▶ int - 2 bytes: de -32.768 à 32.767;
- ▶ float - 4 bytes: de -3.4×10^{-38} à 3.4×10^{38} ;
- ▶ double - 8 bytes: de -3.4×10^{-308} à 3.4×10^{308} .

float — 4 bytes
8 bytes ← double

Atenção

Apenas **float** e **double** podem armazenar valores fracionários!

Linguagem C

Variáveis - Modificadores

- ▶ Para obtermos um tamanho de variável diferente dos tamanhos padrões podemos utilizar dois modificadores: long e short.
 - ▶ Uma variável do tipo **long** deve ser de tamanho MAIOR ou IGUAL a variável do tipo basico modificado.
 - ▶ Uma variável do tipo **short** deve ser de tamanho MENOR ou IGUAL a variável do tipo basico modificado.
 - ▶ Exemplos:
 - ▶ **short int** - 2 bytes: de -32.768 à 32.767;
 - ▶ **int** - 2 bytes: de -32.768 à 32.767;
 - ▶ **long int** - 4 bytes: de -2.147.483.648 à 2.147.483.647;
- long long int*

Linguagem C

Variáveis - Modificadores

- ▶ Todos os tipos básicos apresentados possuem sinal.
- ▶ Se não for necessário o uso de sinal é possível utilizar o modificador unsigned.
- ▶ As variáveis conseguem, com o mesmo espaço, armazenar um valor mais alto.
- ▶ Para garantir que aquela variável tem sinal utilizamos o modificador signed.
- ▶ Exemplos:
 - ▶ **signed int** - 2 bytes: de -32.768 à 32.767;
 - ▶ int - 2 bytes: de -32.768 à 32.767;
 - ▶ unsigned int - 2 bytes: de 0 à 65.535;

Linguagem C

Variáveis - Declaração



► Variáveis:

- Consistem em um tipo seguido de nome da variável.
- Devem ser declaradas antes de iniciar a codificação.
- Variáveis do mesmo tipo podem ser declaradas separadas por vírgula.

PRINT IPAL

Declaração de variáveis

```
1 ⇒ int main(int argc, char *argv[]){  
2     //Declaração das variáveis  
3     int numFuncionarios; *  
4     float salarioMinimo, bonificacao;  
5     double imposto, descontoEmFolha;  
6     return 0;  
7 } //end main
```

#include < >
#define
numFunc *int*
= 
bonificacao *float*
= 

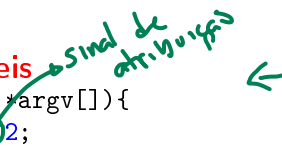
*int a,
b,
c;*

Linguagem C

Variáveis - Inicialização

- Inicializar uma variável é atribuir um valor a esta no momento de sua declaração.

Inicialização de variáveis



```
1 int main(int argc, char *argv[]){  
2     int numFuncionarios = 2;  
3     float salarioMinimo = 510.0;  
4     double imposto = 0.25, descontoEmFolha = 151.97;  
5     printf("O salario minimo eh %f", salarioMinimo);  
6     return 0;  
7 } //end main
```

Linguagem C

Variáveis - Inicialização de conjunto de caracteres

- ▶ Para armazenar textos utilizamos um vetor de caracteres.
- ▶ A inicialização deste vetor pode ser feita utilizando o texto entre aspas duplas.

Armazenamento de texto

```
1 int main(int argc, char *argv[]){  
2     char nome[10] = "Jose";  
3     printf("Meu nome eh %s", nome);  
4     return 0;  
5 } //end main
```



Linguagem C

Variáveis - Atribuição

a ← 10

- ▶ A alteração do valor de uma variável no meio do programa é chamada atribuição.
- ▶ O operador de atribuição é o sinal de igual (=).

Atribuição de variáveis

```
1 int main(int argc, char *argv[]){  
2     float valor;  
3     valor = 1234.56;  
4     printf("Valor sem nota %f", valor);  
5     valor = valor * 1.25;  
6     printf("Valor com nota %f", valor);  
7     return 0;  
8 } //end main
```

valor = 543.20 ^{Float}

Linguagem C

Variáveis - *typecast*

$\text{int} \Leftrightarrow \text{valor inteiro}$ $\text{int } a \times \text{float } b$
 $\text{real} \Leftrightarrow \text{" real}$ int
 $\text{char} \Leftrightarrow \text{" caracter}$

- ▶ **typecast** é a operação de mudança de tipo de um valor.
- ▶ Para realizar um **typecast** basta colocar o tipo desejado entre parênteses na frente do valor a ser convertido.

Atribuição de variáveis

```
1 int main(int argc, char *argv[]){  
2     int valor;  
3     float imposto;  
4     valor = 300; //atribuição  
5     imposto = (int) valor * 0.257; 77.1  
6     printf("Valor = %d, imposto = %f", valor, imposto);  
7     //Valor = 300 e imposto = 77  
8     return 0;  
9 }
```

valor int 300 float 77
imposto

Linguagem C

Variáveis - typecast

Cuidado!

Ao realizar typecast tenha cuidado para não perder informação. No caso anterior, o valor acaba sendo truncado pois, uma variável do tipo int não possui virgula.

void a; a ^{void}

(int) a
(float) a

Linguagem C

Constantes

- ▶ Uma constante possui valor fixo e inalterável.
- ▶ Para declarar uma constante utilizamos a diretiva #define mais o nome da constante e o valor a ser atribuído.

Definição de constantes

#define PI 3.1412
float

```
1 #include<stdio.h>
2 #define DISTANCIA 262
3 int main(int argc, char *argv[]){
4     printf("Distancia entre Itajuba e SP = %d", DISTANCIA);
5     return 0;
6 } //end main
```

#define SEGUNDA 2
#define TERÇA 3

Linguagem C

Operadores

- ▶ A linguagem C tem uma grande quantidade de operadores.
- ▶ Os operadores podem ser divididos em grupos como: aritméticos, aritméticos de atribuição, incremento, decremento, relacionais e lógicos.
- ▶ **Operadores aritméticos:** soma e subtração têm a mesma prioridade, que é menor do que a multiplicação, divisão e resto da divisão.

Operadores

Operador	Descrição	Operador	Descrição
=	Atribuição✓	+	Soma
-	Subtração	*	Multiplicação
/	Divisão	%	Resto da divisão

Linguagem C

Operadores - Operadores Aritméticos

Exemplos de operadores aritméticos

1 `int a = 1, b = 3, c = 5;`

2 `int l, m, n, o, p;`

<code>//código ✖</code>	<code>//interpretação</code>	<code>//resultado ✖</code>
<code>l = 17 % c;</code>	<code>l = 17 % 5;</code>	<code>l = 2;</code>
<code>m = 15 + b;</code>	<code>m = 15 + <u>3</u>;</code>	<code>m = 18;</code>
<code>n = 23 - a;</code>	<code>n = 23 - 1;</code>	<code>n = 22;</code>
<code>o = 18 / b;</code>	<code>o = 18 / 3;</code>	<code>o = 6;</code>
<code>p = 4 * c;</code>	<code>p = 4 * 5;</code>	<code>p = 20;</code>

Linguagem C

Operadores - Operadores Aritméticos de Atribuição

- Pode-se combinar os operadores aritméticos (+, -, *, /, %) com o operador de atribuição da seguinte forma:

Combinação operadores aritméticos e atribuição

```
1 int a = 1, b = 3, c = 5;  
2 int l, m, n, o, p;
```



//contração	//expandido
<u>l</u> * = 4;	<u>l</u> = <u>l</u> * 4;
m /= 2;	<u>m</u> = <u>m</u> / 2;
n += 5;	<u>n</u> = <u>n</u> + 5;
o -= 8;	<u>o</u> = <u>o</u> - 8;
p %= 5;	<u>p</u> = <u>p</u> % 5;

- Este tipo de contração é muito utilizado na linguagem C, pois facilita a escrita.

Linguagem C

Operadores - Operadores de Incremento e Decremento

- ▶ Uma operação muito comum nos programas em C é realizar o incremento ou decremento de uma variável.
- ▶ Em C existem dois operadores específicos:
 - ▶ ++ incrementa de 1 seu operando.
 - ▶ -- decrementa de 1 seu operando.
- ▶ São utilizados para realizar contagens progressivas ou regressivas.
- ▶ Trabalham de dois modos:
 - ▶ pré-fixado: o operador aparece antes do nome da variável.
Exemplo: ++n; onde n é incrementado antes de seu valor ser usado.
 - ▶ pós-fixado: o operador aparece após o nome da variável.
Exemplo: n++, onde n é incrementado depois de seu valor ser usado.

Linguagem C

Operadores - Operadores de Incremento e Decremento

Exemplos Operadores de Incremento e Decremento

1 `int a = 4;`

//comando	//interpretação	//resultado
a ++;	a = a + 1;	a = 5;
a--;	a = a - 1;	a = 3;
l = 2 * a++;	l = 2 * 4; +1	l = 8 9
m = 2 * ++a;	m = 2 * 5;	m = 10;

$$l = \underbrace{2 * 4}_{8} + 1$$

Linguagem C

Operadores - Operadores Relacionais

- ▶ Os operadores relacionais são utilizados para fazer comparações.
- ▶ O resultado de uma comparação deve ser **verdadeiro** ou **falso**.
- ▶ A linguagem C **não** define um tipo lógico.
 - ▶ Se o resultado de uma comparação for **falso**, o resultado da operação será 0 (zero).
 - ▶ Se o resultado de uma comparação for **verdadeiro**, o resultado da operação será 1.

Operadores Relacionais

Operador	Descrição	Operador	Descrição
>	Maior	<	Menor
>=	Maior ou igual	<=	Menor ou igual
<u>==</u>	Igualdade	<u>!=</u>	Diferente

$a=5$ a 5

$a==10$
 $5==10$
 $x=1$
 $x<=9$

$x<10$

Linguagem C

Operadores - Operadores Relacionais

- Programadores de linguagem C definem seu próprio tipo lógico chamado *bool* da seguinte forma:

Definição de *bool*

```
1 typedef enum {false,true} bool;
```

#include
#define

0 *1*

Exemplo operadores relacionais

```
1 #include<stdio.h>  
2 typedef enum {false,true} bool;  
3 int main(int argc, char *argv[]){  
4     int idade = 18;  
5     bool longMaiorIdade = (idade >= 18);  
6     bool idadeVelho = (idade > 50);  
7     printf("Maior idade = %d e velho = %d", longMaiorIdade, idadeVelho);  
8     return 0;  
9 } //end main
```

18 *⇒ TRUE*

FALSE

Resultado: maiorIdade = 1 e velho = 0

Linguagem C

Operadores - Operadores Lógicos

- ▶ Os operadores lógicos são:

- ▶ **Lógico E**: indicado por &&.

- ▶ **Lógico OU**: indicado por ||.

- ▶ **Lógico Negação**: indicado por !.



- ▶ Se exp1 e exp2 são duas expressões simples:

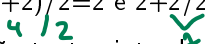
- ▶ exp1 && exp2: é verdadeira se as duas expressões forem verdadeiras.

- ▶ exp1 || exp2: é verdadeira se uma ou as duas expressões forem verdadeiras.

- ▶ ! exp1: é verdadeira se exp1 for falsa e vice-versa.

Linguagem C

Expressões e Comentários

1. **Expressões:** a linguagem C dá ao usuário uma grande liberdade no momento de escrever expressões.
 - ▶ Isto é responsável pela compactação do código escrito em C, mas faz com que as expressões escritas por programadores mais experientes sejam mais complicadas.
 - ▶ Para facilitar a interpretação das expressões é comum o uso de parênteses para evitar a ambiguidade.
 - ▶ **Exemplo:** $(2+2)/2=2$ e $2+2/2=3$

2. **Comentários:** são textos introduzidos no meio do programa fonte com a intenção de torná-lo mais claro.
 - ▶ `//`: usado para comentar uma linha do código.
 - ▶ `/* ... */`: usado para comentar mais de uma linha de código.

Linguagem C

Funções de Entrada e Saída

⇒ ESCREVA

► Função de saída `printf()`.

- Está definida na biblioteca `stdio.h`.
- É uma das funções de entrada e saída e no interior dos parênteses estão as informações passadas na tela do computador.

Sintaxe

`printf`("expressão de controle", lista de argumentos);

- *Expressão de controle*: indica a mensagem que vai ser impressa na tela.
- *Lista de argumentos*: indica os valores que serão passados para a expressão controle.
- É responsável tanto por imprimir as informações na tela do computador, quanto por formatá-las.

Linguagem C

Funções de Entrada e Saída

- As informações apresentadas na tela do computador podem ser formatadas através de códigos especiais:

a=5 escreva("A = ", a) A = 5

Códigos Especiais em C

Códigos especiais	Descrição
<code>\n</code>	Nova linha
<code>\t</code>	Tabulação
<code>\b</code>	Retrocesso
<code>\"</code>	Aspas
<code>\\</code>	Barra
<code>\f</code>	Salta página

*a=5
printf("A = %d\n", a);*

Código printf()

Código printf()	Formato
<code>%c</code>	Caracter simples
<code>%d</code>	Inteiro
<code>%e</code>	Notação Científica
<code>%f</code>	Ponto flutuante
<code>%o</code>	Inteiro octal
<code>%s</code>	Cadeia de caracteres
<code>%u</code>	Decimal sem sinal
<code>%x</code>	Hexadecimal

Linguagem C

Funções de Entrada e Saída

Exemplo 01

```
1  #include<stdio.h>
2  int main(int argc, char *argv[]){
3      printf("Este eh o numero dois: %d", 2);
4      return 0;
5  }//end main
```

Resultado na tela: Este eh o numero dois: 2

Exemplo 02

```
1  #include<stdio.h>
2  int main(int argc, char *argv[]){
3      printf("%s esta a %d milhoes de milhas do Sol", "Venus", 67);
4      return 0;
5  }//end main
```

Resultado na tela: Venus esta a 67 milhoes de anos do Sol

Linguagem C

Funções de Entrada e Saída

- ▶ Os comandos de formato podem ter notificadores que especificam a largura do campo e o número de casas decimais.
- ▶ Um número inteiro é colocado entre o sinal % e o comando de formato age como um especificador de largura mínima do campo.
- ▶ Caso a *string* ou o número seja maior que o mínimo, será totalmente impressa, mesmo que ultrapasse o mínimo.

Exemplos

```
1 //Preenche com espaços em branco
2 printf("%5d", 350); //Tela:.. 350
3 //Preenche com zeros
4 printf("%05d", 350); //Tela:00350
```

Linguagem C

Funções de Entrada e Saída

Exemplos

```
1 //Preenche com espaços em branco
2 printf("%4.2f \n", 3456.78); //Tela: 3456.78
3 printf("%3.1f \n", 3456.78); //Tela: 3456.8
4 printf("%10.3f \n", 3456.78); //Tela: 3456.780
5 //Alinhamento com casas decimais
6 printf("%10.2f %10.2f %10.2f \n", 834.0, 1500.55, 480.21);
7 printf("%10.2f %10.2f %10.2f \n", 23, 4567.64, 9.12);
```

//Resultado na tela

834.00	1500.55	480.21
23.00	4567.64	9.12

- O sinal de menos (-) precedendo a especificação do tamanho do campo após o sinal %, justifica os campos à esquerda.

Exemplos

```
1 printf("%-10.2f %-10.2f %-10.2f \n", 834.0, 1500.55, 480.21);  
2 printf("%-10.2f %-10.2f %-10.2f \n", 23, 4567.64, 9.12);
```

// Resultado na tela

```
834.00    1500.55    480.21  
23.00     4567.64     9.12
```

Linguagem C

Funções de Entrada e Saída

- ▶ Função de saída putchar(): exibe um único caracter na tela.
 - ▶ Está definida na biblioteca stdio.h.
 - ▶ Ao contrário de *strings*, escritas entre aspas, constantes tipo caracter são escritas entre apóstrofo em C.

Sintaxe

putchar(caracter);

- ▶ Exemplo: **putchar**('a');

Linguagem C

Funções de Entrada e Saída

⇒ *leia*

► Função de entrada scanf():

- Está definida na biblioteca stdio.h.
- É o complemento da função **printf()**.
- Permite ler dados formatados da entrada do teclado.
- Sua sintaxe é similar à do **printf()**, ou seja, é uma expressão de controle seguida por uma lista de argumentos separados por vírgula.

Sintaxe

scanf("expressão de controle", lista de argumentos);

- *Expressão de controle*: são os códigos de formatação, precedidos por um sinal de %.
- *Lista de argumentos*: consiste nos endereços das variáveis.

Linguagem C

Funções de Entrada e Saída

- ▶ A linguagem C oferece um operador para tipos básicos chamado **operador de endereço** e referenciado ao símbolo **&** que retorna o endereço.

Código scanf()

Código scanf()	Leitura	Código scanf()	Leitura
%c	Caracter simples	%d	Inteiro
%e	Número notação científica	%o	Inteiro octal
%f	Número de ponto flutuante	%x	Hexadecimal
%s	Cadeia de caracteres	%l	Inteiro longo
%u	Decimal sem sinal		

Handwritten notes:

```
float d;  
int a;  
a =    scanf("%d", &a);    scanf("%.d%.f", &a, &d);
```


Linguagem C

Funções de Entrada e Saída

Exemplo 01

```
1  #include<stdio.h>
2  int main(int argc, char *argv[]){
3      int idade;
4      printf("Digite a sua idade = ");
5      scanf("%d", &idade);
6      printf("Voce tem %d anos.\n", idade);
7      return 0;
8  }//end main
```

Resultado na tela: Voce tem 22 anos.

char nome[30];
scanf("%s", nome);
gets(nome);
↓
std lib.

Linguagem C

Funções de Entrada e Saída

Exemplo

```
1  #include<stdio.h>
2  int main(int argc, char *argv[]){
3      int n;
4      printf("N = ");
5      scanf("%d", &n);
6      printf("Valor de n = %d, no endereco: %u \n", n, &n);
7      return 0;
8  }//end main
```

Resultado na tela: Valor de $n = 2$, no endereco: 3220420448

Linguagem C

Funções de Entrada e Saída

- **Função de entrada `getchar()`**: lê o caracter do teclado e permite que ele seja impresso na tela. Está definida na biblioteca **`stdio.h`**.

Exemplo

```
1  #include<stdio.h>
2  int main(int argc, char *argv[]){
3      char n;
4      printf("Digite um caracter: ");
5      n = getchar();
6      printf("A tecla pressionada foi %c \n", n);
7      printf("Digite outro caracter: ");
8      n = getchar();
9      printf("A tecla pressionada foi %c \n", n);
10     return 0;
11 }
```

Resultado na tela

Digite um caracter: **a**
A tecla pressionada foi: a
Digite outro caracter: **b**
A tecla pressionada foi: b