# Linguagem e Técnicas de Programação


**Tania Basso**

Colégio técnico de limeira - cotil
UNIVERSIDADE ESTADUAL DE CAMPINAS– UNICAMP – BRASIL

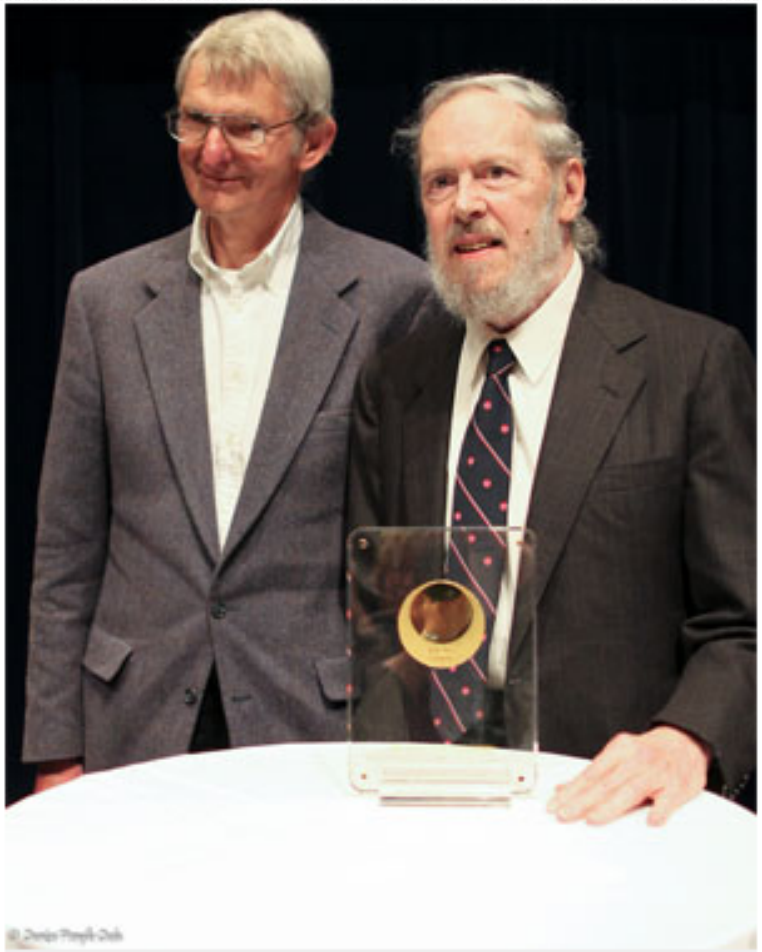
# Linguagem C – Breve Histórico

- instruções consistem em termos semelhantes a expressões algébricas, com palavras chaves em inglês

- foi desenvolvida a partir de duas linguagens anteriores, a BCPL e B, criadas no final da década de 60.

# Linguagem C – Breve Histórico

- foi criada por Dennis Ritchie e Brian W. Kernighan, em 1972, no centro de pesquisa da Bell Laboratories. Sua primeira utilização importante foi a reescrita do sistema Unix. Em meados da década de 70, o Unix foi liberado para utilização nas universidades, dando o impulso para o sucesso da linguagem.



Dennis Ritchie (à dir.), em maio de 2011, recebe o Prêmio Japão pela criação do Unix (Foto: Denise Panyik-Dale/Creative Commons)

# Linguagem C – Breve Histórico

→ANSI C (American National Standards Institute) é a versão da linguagem C padronizada em 1989 através da International Standards Organization — ISO.

→C++ é uma linguagem derivada da linguagem C.

→C++ acrescenta conceitos de orientação a objetos

→Qualquer programa em C compilado no padrão ANSI também pode ser compilado em C++, ou seja, C e C++ são compatíveis.

# Linguagem C – Características

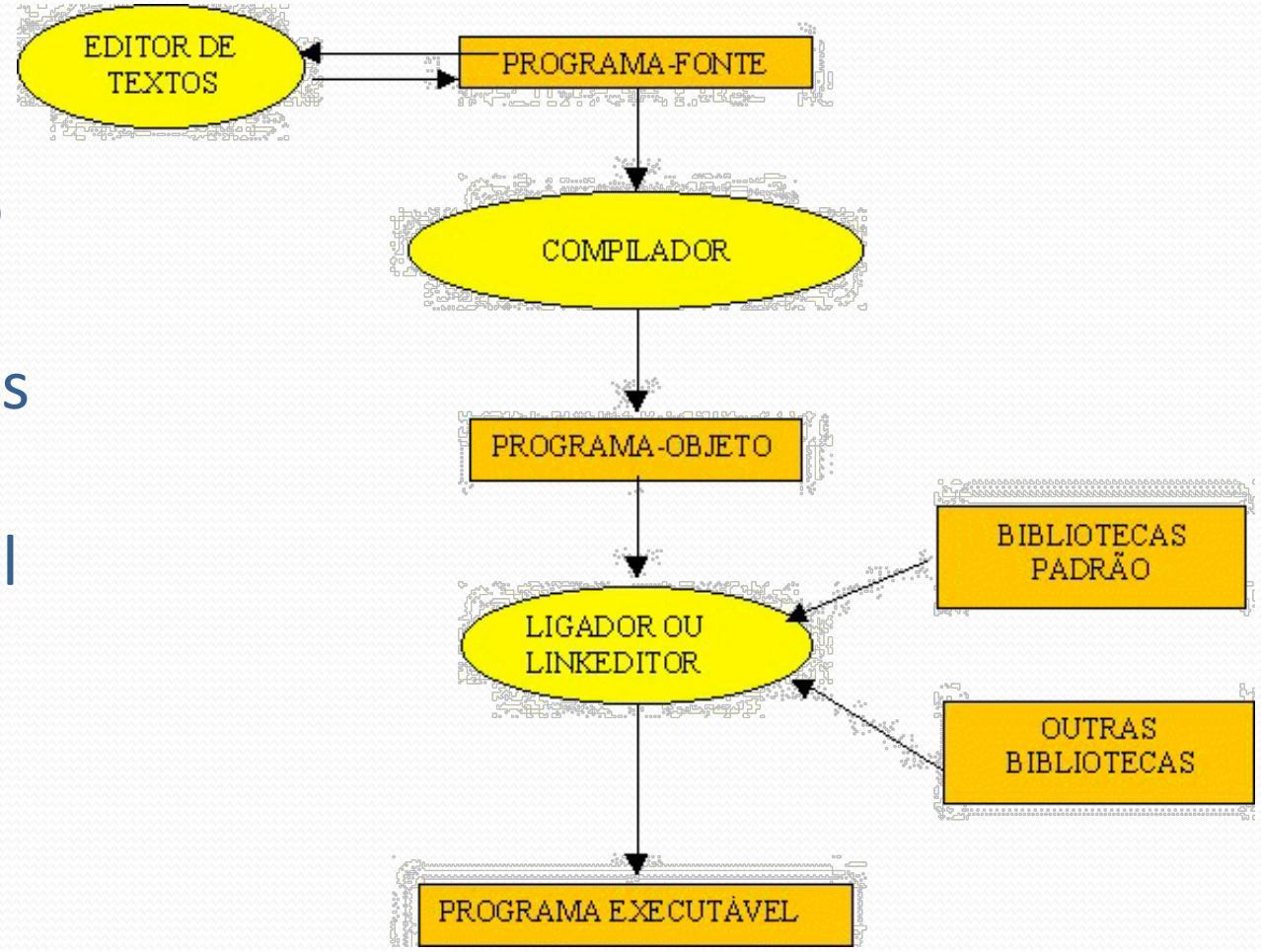
A geração de um programa executável a partir do programa fonte obedece a uma sequência de operações

1.EDITOR – módulo fonte em C. O programador digita um programa em um editor.

2. COMPILADOR – módulo em linguagem de máquina ou código objeto.

# Linguagem C – Características

3. LIGADOR (LINKER) – Faz a ligação do código objeto com o código de funções para produzir um código executável



# Linguagem C – Características

É importante salientar que a linguagem C é sensível ao caso (***case-sensitive***), ou seja, letras maiúsculas e minúsculas são tratadas como caracteres diferentes (por exemplo, a é diferente de A).

# Estrutura Básica de um Programa

```
#include <nome-da-biblioteca>

void main()
{
   declaração-de-variáveis;
   comandos;
}
```

As definições de pré-processamento (diretivas ou arquivos de inclusão) provocam a inclusão de um outro arquivo em nosso programa fonte.

Toda instrução em C termina com um ponto-e-vírgula.

Todo programa em C deve ter uma função chamada main(), que é a primeira função a ser executada. O programa termina sua execução quando é encontrada a chave que fecha o corpo da função main().

Toda função C deve começar com uma chave de abertura de bloco e deve terminar com uma chave de fechamento de bloco.

# Diretivas

Diretiva **include** → Copia o conteúdo do arquivo especificado para dentro do programa. Sintaxe:
# include<nome do arquivo>

→usada para arquivos do tipo .h (header) que contém funções da biblioteca padrão, definição de tipos e macros.

→O arquivo é procurado em um diretório padrão de headers do compilador. (quando se utiliza aspas duplas, ao invés de <>, o arquivo é procurado no diretório atual)

# Diretivas

| Arquivo | Descrição |
|---|---|
| stdio.h | funções de entrada e saída |
| stdlib.h | funções de uso genérico |
| string.h | funções de tratamento de strings |
| math.h | funções matemáticas |

Tabela 1: Alguns arquivos de cabeçalhos.

alocação de memória, controle de processos, conversões e outras

# Diretivas

Diretiva **define**→ substitui toda ocorrência de um nome especificado por um valor determinado (constantes).
Sintaxe:

#define nome valor

Por convenção, todo nome deve ser escrito em letras maiúsculas.
Ex:
#define PI 3.14159

# Palavras reservadas

→possuem um significado especial para o compilador C. O programador não deve utilizá-las como identificadores ou nomes de variáveis. Conjunto completo de palavras reservadas do ANSIC :

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

Tabela 2: Palavras reservadas do C.

# Declaração de variáveis

```
int contador;
float acumulador;
```

declaração

```
int contador = 0;
float tempo = 30.00;
```

Declaração+ inicialização

Tabela 4: Tipos de variáveis em C.

| Tipo | Bit | Bytes | Faixa de Abrangência |
|---|---|---|---|
| char | 8 | 1 | -128 até 127 |
| int | 16 | 2 | -32768 até 32767 |
| float | 32 | 4 | 3.4e-38 até 3.4e+38 |
| double | 64 | 8 | 1.7e-308 até 1.7e+308 |
| void | 0 | 0 | nenhum valor |

E = potência de dez → $3.863x10^9$ é representado como 3.863E+9

# Operadores

**Atribuição** → é o sinal de igual (=). Atribui a expressão da direita à variável à sua esquerda. Em C pode ocorrer atribuições múltiplas.

```
y = 3;
y = x = 3;
y = ( x = 3 );
```

# Operadores

**Aritméticos** → binários (dois operandos) ou unário (um operando)

| | |
|---|---|
| + | Adição |
| - | Subtração |
| * | Multiplicação |
| / | Divisão |
| % | Módulo (resto da divisão do inteiro à sua esquerda pelo inteiro à sua direita) |

# Operadores

**Incremento ou decremento→** Incrementam (++) ou decrementam (–) a variável operando de 1. Podem ser prefixado ou pósfixado.

x = x + 1;
++ x; ou x++; adiciona 1 a x

# Operadores

**Aritmético de atribuição→** Combinam operações aritméticas com atribuições. Compactam as operações. Segue a regra:

variável operação= expressão

Exemplo:

i += 2; equivale a i = i + 2
x *= y + 1 equivale a x = x  (y + 1)
t /= 2.5; equivale a t = t/2.5
p %= 5; equivale a p = p % 5

# Operadores

**Relacionais→** São eles:

> Maior,
>= Maior ou igual
< Menor
<= Menor ou igual,
== igual
! = Diferente

# Operadores

**Lógicos→** São 3 operadores:

| Operador | Descrição |
|---|---|
| && lógico E (*and*) | Resulta 1($V$) somente se as duas expressões forem verdadeiras |
| \|\| lógico ou (*or*) | Resulta 0 ($F$) somente se as duas expressões forem falsas |
| ! lógico não (*not*) | Resulta 1 ($V$) somente se a expressão for falsa |

# Entrada - Saída

Função ***printf()***→ responsável por escrever e formatar os dados de saída. Está definida em stdio.h e tem a seguinte forma:

printf(string de controle de formato , outros argumentos);

A string de controle de formato (que fica entre aspas) descreve o formato da saída; outros argumentos (opcionais) correspondem a cada especificação de conversão na string de controle de formato.

printf (“A soma é: %d “ , soma) ou
printf (“A maior idade é %d e a menor idade é %d”, maior, menor)

# Entrada - Saída

| Especificador | Descrição |
|---|---|
| % d | Exibe um inteiro decimal |
| % i | Exibe um inteiro decimal com sinal |
| % f | Exibe valores de ponto flutuante |
| % e | Exibe valores de ponto flutuante em notação exponencial |
| % c | Exibe caracteres |
| % s | Exibe strings |

# Entrada - Saída

Função ***scanf()***→ entrada de valores numéricos (int, float, double). A formatação dos dados é semelhante à função printf.

Deve-se utilizar obrigatoriamente o símbolo & antes do nome da variável a ser lida.

Está definida em stdio.h.

```
int n;
printf ( ”Informe o número de termos: \n”);
scanf(”%d”, &n);
```

# Entrada - Saída

## Outras funções de entrada de dados

- ***getchar()*** — Lê um caracter e retorna-o. Deve-se teclar ENTER após o caracter digitado. Exemplo:

```
char ch;
ch = getchar();
printf ( ”O caracter digitado foi %c \n”, ch);
```

- ***gets()*** — Lê strings.

```
gets(string digitada);
```

# Um Exemplo

```
#include <stdio.h>
main()
{
    int inteiro1, inteiro2, soma;
    printf (”Entre com o primeiro inteiro\n”);
    scanf (”%d”, &inteiro1);
    printf (”Entre com o segundo inteiro\n”);
    scanf (”%d”, &inteiro2);
    soma = inteiro1 + inteiro2;
    printf (”A soma eh %d\n”, soma);
}
```

# Um Exemplo

```
#include <stdio.h>
main()
{
    int num1, num2, soma;
    printf (”Entre com o primeiro inteiro\n”);
    scanf (”%d”, &num1);
    printf (”Entre com o segundo inteiro\n”);
    scanf (”%d”, &num2);
    soma = num1 + num2;
    printf (”A soma eh %d\n”, soma);
}
```

# Estruturas de seleção

- **comando *if()***

```
if ( condição )
      instrução 1;
else
      instrução 2;
```

- Se condição 1 for verdadeira, é executada a instrução 1; caso contrário, é executada a instrução 2.
- **Múltiplas instruções devem ser delimitadas por chaves em um bloco**.
- O uso do else é opcional.

# Estruturas de seleção

**Exemplo**: Programa para determinar se um número digitado é par ou ímpar

# Estruturas de seleção

```
#include <stdio.h>
int main(void)
{
    int num;
    printf (”Digite um numero:\n”);
    scanf(”%d”, & num);
    if(num%2==0)
    {
        printf(”%d eh par\n”, num);
        printf(”A metade é %d \n”, num/2);
    }
    else
    {
        printf(”%d é impar\n”, num);
    }
    system("PAUSE");
}
```

# Estruturas de seleção

- **comando *switch ()***

```
switch ( valor ) {
        case constante1:
                instruções;
                break;
        case constante2:
                 instruções;
                break;
        case constanteN:
                 instruções;
                break;
        default:
                 instruções;
        }
```

# Estruturas de seleção

- O conteúdo de uma variável (especificada no comando switch*)* é comparado com um valor constante.
- Caso a comparação seja verdadeira, as instruções seguintes ao *case* são executadas.
- Após encontrar o comando *break*, o restante do *switch* é desprezado.
- Caso nenhum valor de *case* coincida com a expressão, então é executada as instruções após *default* (seu uso é opcional)

# Estruturas de seleção

**Exemplo**: Programa que lê dois valores e oferece um menu para o usuário (1- soma, 2- subtração, 3 – multiplicação) desses valores.

```
int main(){
    int codigo, qtd;
    float result, valor1, valor2;
    printf ("Digite o valor 1");  scanf("%f", & valor1);
    printf ("Digite o valor 2");  scanf("%f", & valor2);
    printf ("\nDigite 1 - soma\n 2 - subtração\n 3 - multiplicação");
    scanf("%i", &codigo);
    switch (codigo){
          case 1:
              result = valor1 + valor2;
              break;
          case 2:
              result = valor1 - valor2;
              break;
          case 3:
              result = valor1 * valor2;
              break;
          default: printf ("operação inválida");
        }
    printf ("\nResultado: %.2f", result);
    return 0;
}
```

# Estruturas de Repetição

- comando ***for()***

```
for ( inicialização ; teste ; incremento ) {
        instrução1;
        instruçãon;
}
```

- inicialização : executada uma única vez antes do laço ser iniciado.
- teste: avalia como V ou F.
- incremento: executada imediatamente após a execução do corpo do laço.

# Estruturas de Repetição

- Possibilidade de declaração de variáveis no próprio for.

- Uma variável declarada em um bloco não é visível fora dele.

- Caso tenha-se mais de uma instrução, delimitar o bloco através de chaves.

- Laços for aninhados (um for dentro de outro).

# Estruturas de Repetição

- Exemplo: Fazer um programa que calcule a soma de todos os números inteiros menores ou igual a um limite dado

# Estruturas de Repetição

```
#include <stdio.h>
int main(){
    int i, limite, soma;
    printf ("Digite o limite: ");
    scanf ("%i", & limite);
    soma = 0;
    for (i=1; i<= limite; i++){
       soma = soma + i;
    }
    printf ("\n\Soma dos inteiros menor ou igual a %i é %i\n\n",
limite, soma);
    system("PAUSE");
}
```

# Estruturas de Repetição

- comando ***while()***

Apropriado em situações onde o laço pode ser terminado inesperadamente, por condições desenvolvidas dentro do laço. Sintaxe:

```
Expressão de inicialização;
while (teste)
{
    instruções;
    expressão de incremento;
}
```

# Estruturas de Repetição

- Exemplo: Fazer um simples programa que faça a contagem de números de 1 a 10

# Estruturas de Repetição

```
#include <stdio.h>
#include <stdlib.h>
main()
{
        int i=0;
        while(i < 10)
        {
                printf ("%d", i);
                i ++;
        }
        system("PAUSE");
}
```

# Estruturas de Repetição

- comando ***do-while()***

Utilizado em situações em que é necessário executar o corpo do laço uma primeira vez e depois avaliar a expressão de teste e criar um ciclo repetido. Sintaxe:

```
do
{
      instrução1;
      instruçãon;
} while (teste);
```

# Estruturas de Repetição

- Exemplo: Fazer o cálculo da média aritmética de uma quantidade indeterminada de valores inteiros e positivos usando, primeiro, o comando *while*. Depois, faça o mesmo programa usando o comando *do-while*. A condição de parada é que o valor digitado seja -1

```c
#include <stdio.h>
#include <stdlib.h>
main()
{
    int valor, cont, soma;
    float media;
    cont=0;
    soma = 0;
    printf ("digite o valor: ");
    scanf ("%d", &valor);
    while (valor > -1){
        soma = soma + valor;
        cont ++;
        printf ("digite o valor: ");
        scanf ("%d", &valor);
    }
    media = soma / cont;
    printf ("Média: %.2f", media);
    system("PAUSE");
}
```

```c
#include <stdio.h>
#include <stdlib.h>
main()
{
    int valor, cont, soma;
    float media;
    cont=0;
    soma = 0;
    do {
        printf ("digite o valor: ");
        scanf ("%d", &valor);
        soma = soma + valor;
        cont ++;
    }while (valor > -1);
    media = soma / cont;
    printf ("Média: %.2f", media);
    system("PAUSE");
}
```

**Cuidado!!!**

```c
#include <stdio.h>
#include <stdlib.h>
main()
{
    int valor, cont, soma;
    float media;
    cont=0;
    soma = 0;
    printf ("digite o valor: ");
    scanf ("%d", &valor);
    while (valor > -1){
        soma = soma + valor;
        cont ++;
        printf ("digite o valor: ");
        scanf ("%d", &valor);
    }
    media = soma / cont;
    printf ("Média: %.2f", media);
    system("PAUSE");
}
```

```c
#include <stdio.h>
#include <stdlib.h>
main()
{
    int valor, cont, soma;
    float media;
    cont=0;
    soma = 0;
    do {
        printf ("digite o valor: ");
        scanf ("%d", &valor);
        if (valor > -1){
            soma = soma + valor;
            cont ++;
        }
    }while (valor > -1);
    media = soma / cont;
    printf ("Média: %.2f", media);
    system("PAUSE");
}
```

# Obrigada

**Contato**

taniabasso@gmail.com