



PROGRAMAÇÃO ESTRUTURADA

TCC00347 (A1)

Terças e Quintas

16 às 18 horas

INTRODUÇÃO

- Estrutura de um programa em C
 - Um programa em C sempre começa a sua execução a partir da função main

```
int main(void) {
```

```
    /* Sequência de Comandos */
```

```
    return 0;
```

```
}
```

ESTRUTURA DE UM PROGRAMA EM C

- A estrutura de um programa em C, pode ser diagramada da seguinte forma:

Inclusão de Bibliotecas

Definição de Constantes

Definição de Funções Auxiliares

Definição da Funções main

Comentários – podem estar em toda parte

EXEMPLO

```
#include <stdio.h>

/* Programa que converte a temperatura de F para C */

int main (void) {

    float grauC, grauF;

    printf("Digite a temperatura em Farenheit:");

    scanf("%f", &grauF);

    grauC = (grauF - 32) / 9 * 5;

    printf("A temperatura em Celsius é: %f", grauC);

    return 0;

}
```

EXEMPLO

Onde são incluídas as bibliotecas.
Ex: stdio.h, math.c
string.h stdlib.h

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

```
/* Programa que converte a temperatura de F para C */
```

```
int main (void) {
```

```
    float grauC, grauF;
```

```
    printf("Digite a temperatura em Farenheit:");
```

```
    scanf("%f", &grauF);
```

```
    grauC = (grauF - 32) / 9 * 5;
```

```
    printf("A temperatura em Celsius é: %f", grauC);
```

```
    return 0;
```

```
}
```

EXEMPLO

Onde são incluídas as bibliotecas.
Ex: stdio.h, math.c
string.h stdlib.h

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

```
/* Programa que converte a temperatura de F para C */
```

Comentário

```
int main (void) {
```

```
    float grauC, grauF;
```

```
    printf("Digite a temperatura em Farenheit:");
```

```
    scanf("%f", &grauF);
```

```
    grauC = (grauF - 32) / 9 * 5;
```

```
    printf("A temperatura em Celsius é: %f", grauC);
```

```
    return 0;
```

```
}
```

EXEMPLO

Onde são incluídas as bibliotecas.
Ex: stdio.h, math.c
string.h stdlib.h

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

Comentário

```
/* Programa que converte a temperatura de F para C */
```

```
int main (void) {
```

Função Principal

```
    float grauC, grauF;
```

```
    printf("Digite a temperatura em Farenheit:");
```

```
    scanf("%f", &grauF);
```

```
    grauC = (grauF - 32) / 9 * 5;
```

```
    printf("A temperatura em Celsius é: %f", grauC);
```

```
    return 0;
```

```
}
```

NOMES DE IDENTIFICADORES

- Nomes de variáveis, constantes e funções
- Os identificadores em C são tratados de forma diferente quando são usadas maiúsculas e minúsculas

```
int main (void) {  
    float grauC, grauF;  
    printf("Digite a temperatura em Farenheit:");  
    scanf("%f", &grauF);  
    grauC = (grauF - 32) / 9 * 5;  
    printf("A temperatura em Celsius é: %f", grauC);  
    ...  
}
```


NOMES DE IDENTIFICADORES

- Nomes de variáveis, constantes e funções
- Os identificadores em C são tratados de forma diferente quando são usadas maiúsculas e minúsculas

```
int main (void) {
```

```
    float grauC, grauF;
```

```
    printf("Digite a temperatura em Farenheit:");
```

```
    scanf("%f", &grauF);
```

```
    grauC = (grauF - 32) / 9 * 5;
```

```
    printf("A temperatura em Celsius é: %f", grauC);
```

```
    ...
```

Aqui está o
problema!

VARIÁVEIS

- Variável é um espaço reservado na memória do computador para armazenar um tipo de dado.
- Devem receber nomes para poderem ser referenciadas e modificadas quando necessário.
- Toda variável tem:
 - um nome (identificador)
 - um tipo de dado
 - um valor
- Restrição para nomes: não é permitido começar o nome com um algarismo (0-9), alguns caracteres não são válidos (*, -, /, +, ...), e palavras reservadas não podem ser utilizadas (main, if, while, ...)

VARIÁVEIS

- É necessário informar o nome e o tipo das nossas variáveis:
 - O compilador precisa saber o tipo do dado para reservar o espaço de memória definido para aquele tipo (quantidade de *bytes*).
 - O nome será usado para representar o espaço que foi reservado.
- Exemplo:

```
int main (void) {  
    float grauC, grauF;  
    ...  
}
```

TIPOS DE DADOS

- Existem poucos tipos de dados básicos em C
 - char: guarda um caractere
 - int: guarda um inteiro
 - float, double: guardam números reais
- Um tipo básico pode ter seu significado alterado por um modificador:
 - short, long, signed, unsigned
- Não existe tipo lógico em C

TIPOS DE DADOS

Tipo	Tamanho	Representatividade
char	1 byte	-128 a 127
unsigned char	1 byte	0 a 255
short int	2 bytes	-32768 a 32767
unsigned short int	2 bytes	0 a 65535
long int	4 bytes	-2.147.483.648 a 2.147.483.647
unsigned long int	4 bytes	0 a 4.294.967.295
int	Depende da máquina	
float	4 bytes	- 10^{-38} a 10^{38}
double	8 bytes	- 10^{-308} a 10^{308}

DECLARAÇÃO DE VARIÁVEIS

- Todas as variáveis devem ser declaradas antes de serem usadas
- Uma declaração especifica um tipo, e é seguida por uma lista de uma ou mais variáveis daquele tipo

```
int inicio, fim;
```

```
float media;
```

```
char c, linha[100];
```

DECLARAÇÃO

- Declaração de constantes:

```
#define PI 3.141516
```

```
#define MIL 1000
```

- Uma variável pode ser inicializada na sua declaração:

- `int inicio = 0;`
- `char letra = 'a';`

ESCOPO DAS VARIÁVEIS

- Variáveis podem ser LOCAIS ou GLOBAIS
 - Uma variável **local** só é reconhecida dentro do bloco onde ela foi declarada
 - Vive enquanto a função está sendo executada
 - Nenhuma outra função tem acesso a ela
 - Para uma melhor prática de programação, as variáveis **locais** devem ser declaradas no início de uma função

ESCOPO DAS VARIÁVEIS

- Variáveis podem ser LOCAIS ou GLOBAIS
 - Uma variável **global** é reconhecida no programa inteiro
 - Definidas no início do código fonte
 - Acessadas por quaisquer funções
 - Existem permanentemente e retêm seus valores armazenados

EXEMPLO

```
int A, B;  
  
int teste ( ) {  
    int C, D;  
    ...  
}  
  
int main (void) {  
    int count;  
    ...  
}
```

EXEMPLO

```
int A, B;
```

GLOBAIS



```
int teste ( ) {
```

```
    int C, D;
```

```
    ...
```

```
}
```

```
int main (void) {
```

```
    int count;
```

```
    ...
```

```
}
```

LOCAIS

OPERADORES

○ Aritméticos:

- + → Adição
- - → Subtração
- * → Multiplicação
- / → Divisão
- % → Resto (Só pode ser usado com inteiro)

○ De Atribuição:

- =

EXEMPLOS

```
int a, b, c;
```

```
a = 5;
```

```
b = 2;
```

```
c = a + b;
```

$c \rightarrow 7$

```
c = a - b;
```

$c \rightarrow 3$

```
c = a * b;
```

$c \rightarrow 10$

```
c = a / b;
```

$c \rightarrow 2$

```
c = a % b;
```

$c \rightarrow 1$

```
float x, y, z;
```

```
x = 5;
```

```
y = 2;
```

```
z = x + y;
```

$z \rightarrow 7.0$

```
z = x - y;
```

$z \rightarrow 3.0$

```
z = x * y;
```

$z \rightarrow 10.0$

```
z = x / y;
```

$z \rightarrow 2.5$

```
z = x % y;
```

Erro!

EXERCÍCIO

- Considere:

```
int a;
```

```
float b, c;
```

```
...
```

```
a = 5/2;
```

```
b = 5/a;
```

```
c = 1.5 + b;
```

- Quais valores terão as variáveis `a`, `b` e `c`?

EXERCÍCIO

- Considere:

```
int a;
```

```
float b, c;
```

```
...
```

```
a = 5/2;
```

```
b = 5/a;
```

```
c = 1.5 + b;
```

- Quais valores terão as variáveis `a`, `b` e `c`?

RESPOSTA:

`a` → 2

`b` → 2.0

`c` → 3.5

OPERADORES COMPOSTOS

Expressões de atribuição:

Considere que inicialmente
`cont = 0;`

<code>+=</code>	soma e atribuição	<code>cont += 4;</code>	<code>cont = cont + 4;</code>	<code>cont → 4</code>
<code>-=</code>	subtração e atribuição	<code>cont -= 2;</code>	<code>cont = cont - 2;</code>	<code>cont → 2</code>
<code>*=</code>	multiplicação e atribuição	<code>cont *= 3;</code>	<code>cont = cont * 3;</code>	<code>cont → 6</code>
<code>/=</code>	divisão e atribuição	<code>cont /= 2;</code>	<code>cont = cont / 2;</code>	<code>cont → 3</code>

OPERADORES

- Incremento e Decremento

`++` → operador de incremento, soma 1 ao operando

`--` → operador de decremento, subtrai 1 do operando

- Os dois operadores podem ser pré-fixados (antes da variável) ou pós-fixados (depois da variável)

- Exemplo: `int n;`

`n++;`

`++n;`

INCREMENTO E DECREMENTO

- Suponha as expressões, onde $n = 5$

- $x = n++;$

- x recebe o valor de n , e depois n é incrementado

- então $x = 5$ e $n = 6$

- $x = ++n;$

- x recebe o valor de n , já incrementado

- então $x = 6$ e $n = 6$

INCREMENTO E DECREMENTO

- Os operadores de incremento ($++$) e decremento ($--$) só podem ser aplicados a variáveis e não a uma expressão.

$(i + j)++;$ \rightarrow expressão inválida!

EXEMPLOS

- A linguagem C oferece diversas formas compactas para se escrever um determinado comando:

```
a = a + 1;
```

```
a++;
```

```
++a;
```

```
a += 1;
```

EXEMPLOS

- Expressões de atribuição:

`i += 2;`

$\rightarrow i = i + 2;$

`x /= y;`

$\rightarrow x = x / y;$

`x *= y + 1;`

$\rightarrow x = x * (y + 1);$ CERTO!

$\rightarrow x = x * y + 1;$ ERRADO!

OPERADORES RELACIONAIS

==

!=

>

>=

<

<=

- Operadores relacionais possuem menor precedência que os aritméticos, logo:

`i < lim - 1;` → **equivale a** `i < (lim - 1);`

OPERADORES LÓGICOS

& & → E

| | → OU

! → Não

A	B
1	1
1	0
0	1
0	0

A & & B
1
0
0
0

A B
1
1
1
0

!A
0
0
1
1

OPERADORES LÓGICOS

- Os operadores lógicos são avaliados da esquerda para a direita e a avaliação termina assim que a veracidade ou falsidade da sentença for reconhecida.

```
a = 3;
```

```
b = 5;
```

```
(a > b) && (b == 5)
```

```
! (a == 3 && b == 5) || (a < b)
```


CONVERSÃO DE TIPO

- Exemplos de algumas conversões automáticas:

```
int a;
```

```
float y;
```

```
a = 9;
```

```
y = a; → y = 9.0
```

→ tipo “menor” atribuído ao “maior”

```
y = 3.5;
```

```
a = y; → a = 3
```

→ tipo “maior” é atribuído ao “menor” e ocorre perda da informação

CONVERSÃO DE TIPOS

- As conversões aritméticas implícitas funcionam da seguinte forma.
- Quando a expressão tiver operandos de tipos diferentes, o menor é convertido para o maior antes da operação
- A operação é feita na precisão do tipo mais representativo

CONVERSÃO DE TIPOS

○ Exemplo:

```
int a;
```

```
double b, c;
```

```
a = 3.5;      → a = 3
```

```
b = a/2.0;    → b = 1.5
```

```
c = 1/3 +b;    → c = 0 + 1.5 → c = 1.5
```

```
c = 1/3.0 +b;  → c = 0.333 + 1.5 → c = 1.833
```

CONVERSÃO DE TIPOS

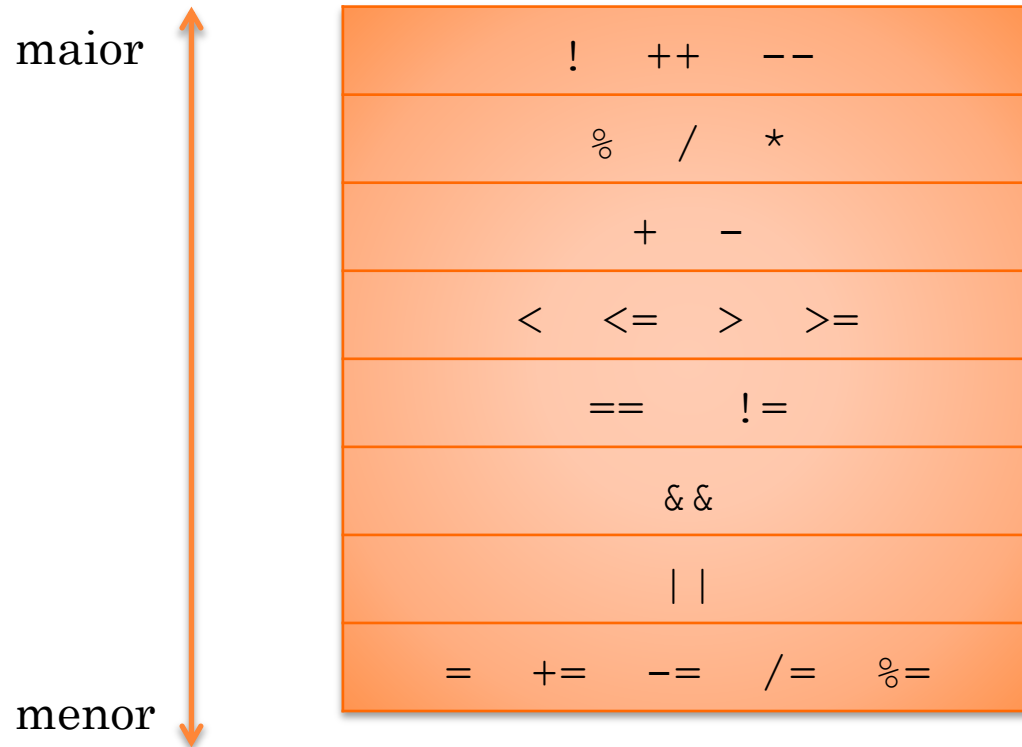
- O programador pode explicitamente requisitar uma conversão de tipo

```
int a, b;
```

```
a = (int) 3.5;
```

→ Não existe aviso (warning)

PRECEDÊNCIA DE OPERADORES



maior	↑	! ++ --
		% / *
		+ -
		< <= > >=
		== !=
		& &
menor	↓	= += -= /= %=

EXERCÍCIOS

○ Considere que $x = 3$ e $y = 6$

○ Responda se a expressão é verdadeira ou falsa

1) $10 > 5 \ \&\& \ ! (10 < 9) \ || \ (3 \leq 2 + 1)$

2) $! (x - 2) < y + 3 \ \&\& \ y - x * 2$

○ Dê os valores finais de x e y em cada expressão:

1) $x *= 3;$

2) $y += (x - y);$

3) $x = ++y;$

EXERCÍCIOS

- Considere:

```
int x, y, z;  
int teste;  
x = 5;  
y = x++;  
z = x--;
```

- Informe os valores de `teste`, `x`, `y` e `z` depois da avaliação das seguintes expressões:

- (a) `teste = !y == !x;`
- (b) `teste = ((x++ > y) || (--z <= y));`
- (c) `teste = ((!x) || (!(!z)));`
- (d) `teste = (((x + y) > z) && (x++));`