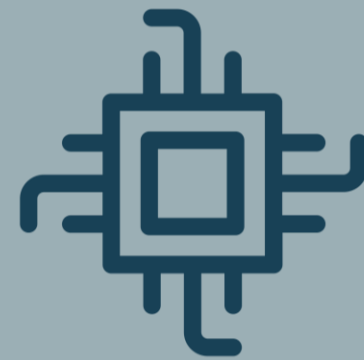




Universidade Federal  
de Campina Grande

# REVISÃO DA LINGUAGEM C



Prof.

**Rafael  
Lima**



# REVISÃO RÁPIDA DE C

- Palavras reservadas em ANSI C

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

- Identificadores (nomes de variáveis, funções etc) somente podem ser constituídos de caracteres **numéricos**, **alfanuméricos** ou “\_”. Não podem ser iniciadas por números. Sensível a caixa alta!

# VARIÁVEIS e TIPOS DE DADOS

- Tipos básicos de dados disponíveis, por padrão, no Atmel Studio 7

Tipo	Nº de bits	Faixa de valores
uint8_t, <b>char</b>	8	0 a 255
int8_t	8	-128 a 127
uint16_t, <b>unsigned int</b>	16	0 a 65535
int16_t, <b>int</b>	16	-32768 a 32767
uint32_t, <b>unsigned long</b>	32	0 a 4294967295
int32_t, <b>long</b>	32	-2147483648 a 2147483647
uint64_t <b>unsigned long long</b>	64	0 a $1,8 \cdot 10^{19}$
int64_t, <b>long long</b>	64	$-9,2 \cdot 10^{18}$ a $9,2 \cdot 10^{18}$
<b>float</b>	32	$\pm 1,18 \cdot 10^{-38}$ a $\pm 3,39 \cdot 10^{+38}$

## Modificador

“**const**”: variável não pode ser alterada em tempo de execução

# VARIÁVEIS e TIPOS DE DADOS

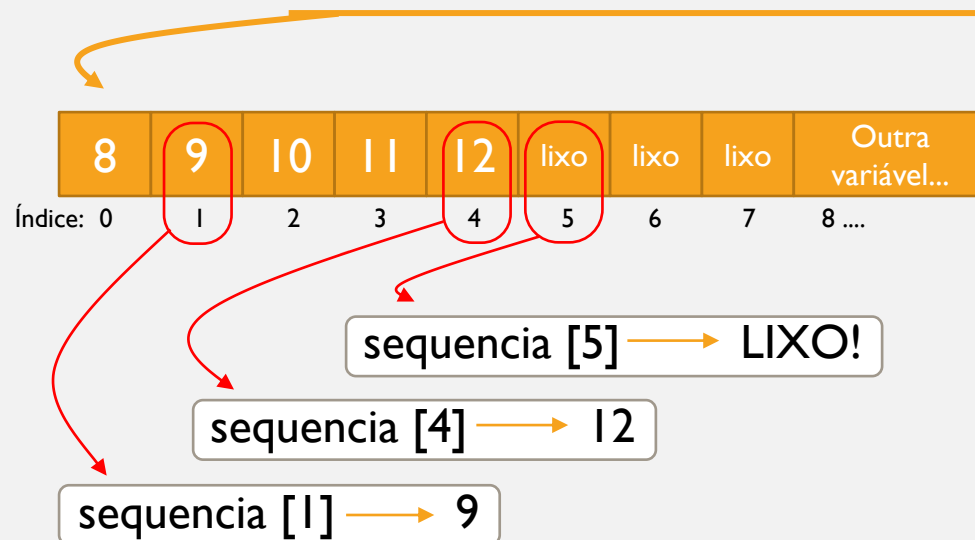
- **Arrays de dados**

Tipo nome\_do\_array [tamanho];

**Ex:** `int8_t` sequencia [5];

Tipo nome\_do\_array [tamanho] = {lista de valores};

**Ex:** `int8_t` sequencia [5] = {8,9,10,11,12};



**O índice inicia em ZERO**

**O fim do array NÃO é  
checado pelo compilador!**  
Cuidado para não sobrescrever  
regiões da memória

# VARIÁVEIS e TIPOS DE DADOS

- Arrays Multidimensionais**

Tipo nome\_da\_matriz [tamanho dim1] [tamanho dim2].. [tamanho dimN];

**Ex:** `int8_t` matriz [2][3];

Tipo nome\_da\_matriz [tamanho dim1]..[tamanho dimN] = {lista de valores};

**Ex:** `int8_t` matriz [2][3] = {{8,9,10}, {11,12,13}};

0	8	9	10	lixo	lixo	Outra variável...
1	11	12	13	lixo	lixo	Outra variável...
	0	1	2	3	4	....

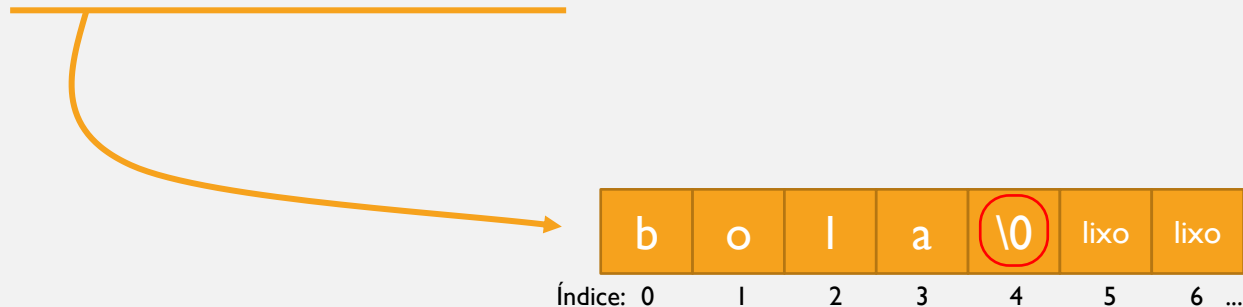
sequencia [1][4] → LIXO!

sequencia [0][2] → 10

# VARIÁVEIS e TIPOS DE DADOS

- **Strings**

- É um array de caracteres
- São terminadas pelo caractere nulo '\0'
- O tamanho total da matriz deve ser 1 caractere maior do que a quantidade a ser armazenada (lembrar que o \0 é inserido automaticamente)
- **Ex:** `char teste [5] = "bola";`



**Caracteres:** 'x'  
**Strings:** "xis"

# VARIÁVEIS e TIPOS DE DADOS

- **Structs**

- É um agrupamento de variáveis sob um mesmo nome.

```
struct [Nome da struct]
{
    Definição de membro;
    Definição de membro;
    ...
    Definição de membro;
} [Nome das instâncias];
```

**Ex:**

```
struct Livro
{
    char titulo[50];
    char autor[50];
    char assunto[100];
    int id;
} livroX;
```

```
struct Livro livroY;
```

```
strcpy( livroX.titulo, "The Fellowship of the Ring");
strcpy( livroX.autor, "J. R. R. Tolkien");
```

```
strcpy( livroY.titulo, "The Hitchhiker's G. to the Galaxy");
strcpy( livroY.autor, "Douglas Adams");
```

# VARIÁVEIS e TIPOS DE DADOS

- **typedef**

- Palavra chave para renomear um tipo

```
typedef tipo_base novo_tipo;
```

**Ex:**

```
typedef struct Livro
```

```
{
```

```
    char titulo[50];
```

```
    char autor[50];
```

```
    char assunto[100];
```

```
    int id;
```

```
} Livro;
```

```
Livro livroX, livroY;
```

```
strcpy( livroX.titulo, "The Fellowship of the Ring");
```

```
strcpy( livroX.autor, "J. R. R. Tolkien");
```

```
strcpy( livroY.titulo, "The Hitchhiker's G. to the Galaxy");
```

```
strcpy( livroY.autor, "Douglas Adams");
```



# VARIÁVEIS e TIPOS DE DADOS

- **enum**

- Conjunto de valores inteiros representados por identificadores

```
enum nome {  
    elem0 = 0,  
    elem1 = 1,  
    ...  
    elemN = N  
};
```

**Ex:**

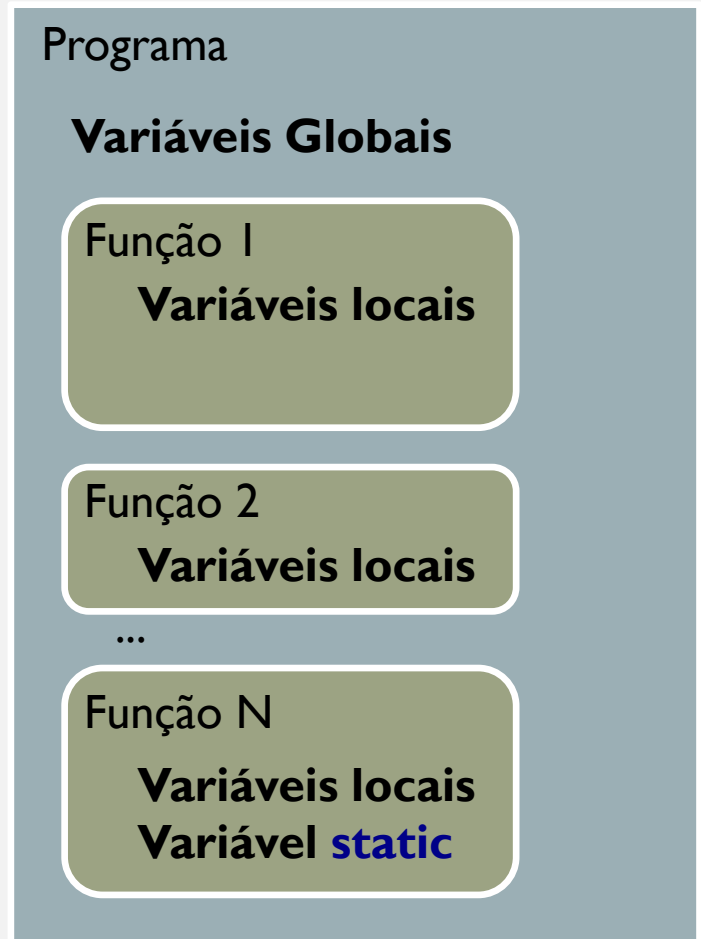
```
enum Cores  
{  
    amarelo = 0,  
    vermelho = 1,  
    azul = 2  
};  
  
enum Cores minhaCor;  
  
minhaCor = azul;  
printf("%d", minhaCor);
```

**saída: 2**

# VARIÁVEIS e TIPOS DE DADOS

- Escopo das variáveis
  - **Variáveis Globais:** variáveis declaradas no corpo principal do programa, fora de qualquer função, podem ser acessadas de qualquer ponto do programa.
  - **Variáveis Locais:** variáveis declaradas dentro de uma função somente podem ser acessadas no interior dessa função (só existem enquanto a função é executada)

**Modificador “static”:** são variáveis locais, porém só são inicializadas uma vez e seu valor é mantido em múltiplas chamadas de função



# VARIÁVEIS e TIPOS DE DADOS

- Representação de constantes em diferentes **bases numéricas**

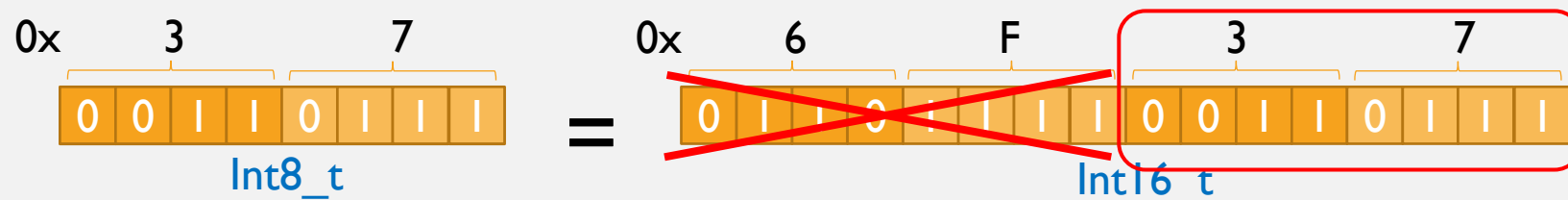
Valor	Base Numérica
<b>99</b>	Decimal
<b>099</b>	Octal
<b>0x99</b>	Hexadecimal
<b>0b10011001</b>	Binário

## VARIÁVEIS e TIPOS DE DADOS

- **Conversão de tipos:** faz parte do padrão ANSI C para lidar com expressões de tipos diferentes. **Regras:**

- Em uma atribuição, o tipo do dado resultante da expressão, do lado direito da igualdade, é convertido no tipo do dado da variável que recebe a atribuição

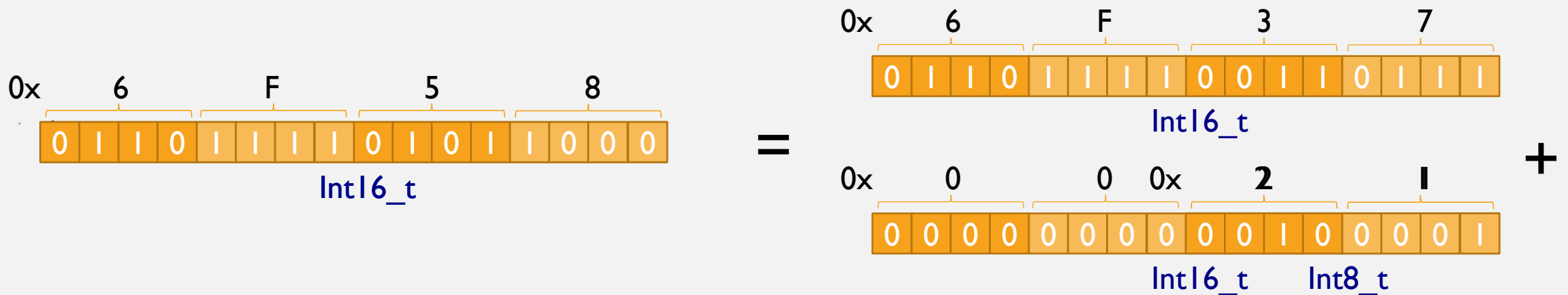
**Ex:** `int8_t = int16_t` ... Serão perdidos os 8 bits mais significativos!



# VARIÁVEIS e TIPOS DE DADOS

- Promoção de tipos: cada par de operando de tipos diferentes é convertido no tipo superior

**Ex:** `int16_t = int8_t + int16_t` ... O tipo `int8_t` é promovido para `int16_t`




## VARIÁVEIS e TIPOS DE DADOS

- A operação de CAST converte explicitamente um tipo em outro. Basta especificar o novo tipo entre parênteses

### Exemplo A:


```
uint8_t    x=0xFF, y=0x01;  
uint16_t   z;  
z = x + y;
```



**0x0000**

### Exemplo B:

```
uint8_t    x=0xFF, y=0x01;  
uint16_t   z;  
z = (uint16_t)x + (uint16_t)y;
```



**0x0100**

# DIRETRIZES DE PRÉ-PROCESSAMENTO

Comando	Descrição	Exemplo
<code>#include &lt;arquivo&gt;</code>	Incluir um arquivo no programa	<code>#include &lt;stdio.h&gt;</code>
<code>#define MARCADOR valor</code>	Substituir um marcador por uma constante	<code>#define PI 3.14</code>
<code>#undef MARCADOR</code>	Remove a definição	<code>#undef PI</code>
<code>#ifdef MARCADOR</code> <code>// código</code> <code>#endif</code>	O código é incluído se Marcador tiver sido definido	<code>#ifdef PI</code> <code>// código...</code> <code>#endif</code>
<code>#ifndef MARCADOR</code> <code>// código</code> <code>#endif</code>	O código é incluído se Marcador NÃO tiver sido definido	<code>#ifndef PI</code> <code>// código...</code> <code>#endif</code>
<code>#if expressão</code> <code>// código se TRUE</code> <code>#elif expressão l</code> <code>// código se TRUE</code> <code>#else</code> <code>// código se FALSE</code> <code>#endif</code>	O código é incluído se a condição for satisfeita	<code>#if X==2</code> <code>cont = 1;</code> <code>#elif X==3</code> <code>cont = 2;</code> <code>#else</code> <code>cont = 3;</code> <code>#endif</code>

# OPERADORES

- **Aritméticos:**

Operador	Ação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto
++	Incremento
--	Decremento



# OPERADORES

- **Relacionais:**

Operador	Ação
>	Maior que
>=	Maior ou igual
<	Menor que
<=	Menor ou igual
==	Igual
!=	Diferente

# OPERADORES

- Lógicos booleanos:

Operador	Ação
<b>&amp;&amp;</b>	AND
<b>  </b>	OR
<b>!</b>	NOT

## Exemplo A:

```
0b010 → V
&& 0b011 → && V
-----
0b001 ← V
```

- Lógicos bit a bit:

Operador	Ação
<b>&amp;</b>	AND
<b> </b>	OR
<b>^</b>	XOR
<b>~</b>	NOT
<b>&gt;&gt;</b>	Desloc. a direita
<b>&lt;&lt;</b>	Desloc. a esquerda

## Exemplo B:

```
0b 0 1 0
& 0b 0 1 1
-----
0b 0 1 0
```

# OPERADORES

- **Memória**

Operador	Ação
<b>&amp;</b>	Endereço do operando
<b>*</b>	Conteúdo do endereço apontado pelo operando

## Exemplo:

End. Memória

14	a = 55
15	b = 32
16	c = 25
17	d = 14
...	...

```
uint8_t a = 55, b = 32, c = 25, d = 14;
```

```
uint8_t *end_a, *end_b;
```

```
printf("%d",a); // 55
```

```
printf("%d",b); // 32
```

```
printf("%d",&a); // 14
```

```
printf("%d",&b); // 15
```

```
end_a = &a;
```

```
end_b = &b;
```

```
printf("%d",end_a); // 14
```

```
printf("%d",end_b); // 15
```

```
printf("%d",*end_a); // 55
```

```
printf("%d",*end_b); // 32
```

**O NOME** de um array é um ponteiro para o seu primeiro elemento!

# OPERADORES

- Atribuição**

Operador	Ação
=	Atribuição simples
+=	Adiciona e atribui
-=	Subtrai e atribui
*=	Multiplica e atribui
/=	Divide e atribui
%=	Resto e atribui
<<=	Desloc. a esquerda e atribui
>>=	Desloc. a direita e atribui
&=	AND bit-a-bit e atribui
^=	XOR bit-a-bit e atribui
=	OR bit-a-bit e atribui

# OPERADORES

- **Outros**

Operador	Ação
<b>?</b>	Ternário
<b>,</b>	Separador de expressões
<b>.</b>	Separador de structs
<b>-&gt;</b>	Ponteiro de elemento de structs
<b>sizeof()</b>	Tamanho da variável

# OPERADORES

- Precedência de operadores

() [] -> .	0
! ~ ++ -- (type) * & sizeof	1
* / %	2
+ -	3
<< >>	4
<<= >>=	5
== !=	6
&	7
^	8
	9
&&	10
	11
?:	12
= += -= *= /= %= >>= <<= &= ^=  =	13
,	14

**MAIOR**



**MENOR**

## MÁSCARAS DE BITS

- **& AND bit a bit:** usado para limpar bits, colocar em 0
- **| OR bit a bit:** usado para ativar bits , colocar em 1
- **^ XOR bit a bit:** usado para trocar o estado dos bits

**Exemplo:**

```
  0b01001000
& 0b11110111
-----
  0b01000000
```

```
  0b01001000
| 0b00000010
-----
  0b01001010
```

```
  0b01001000
^ 0b00000001
-----
  0b01001001
```

# DECLARAÇÕES DE CONTROLE

- **Declarações de teste condicional:**

- **Comando If - else**

**If** (condição)

{

//Executa comandos se condição for verdadeira

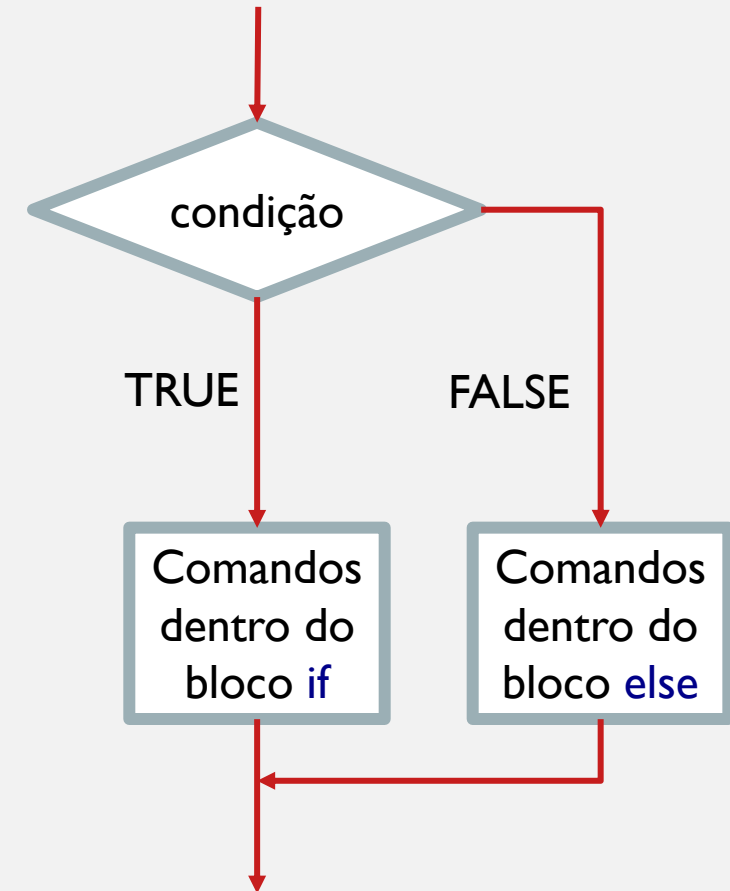
}

**else**

{

//Executa comandos se condição for falsa

}





# DECLARAÇÕES DE CONTROLE

- **Comando Switch**

**switch** (variavel)

{

**case** const1:

comandos...

**break;**

**case** const2:

comandos...

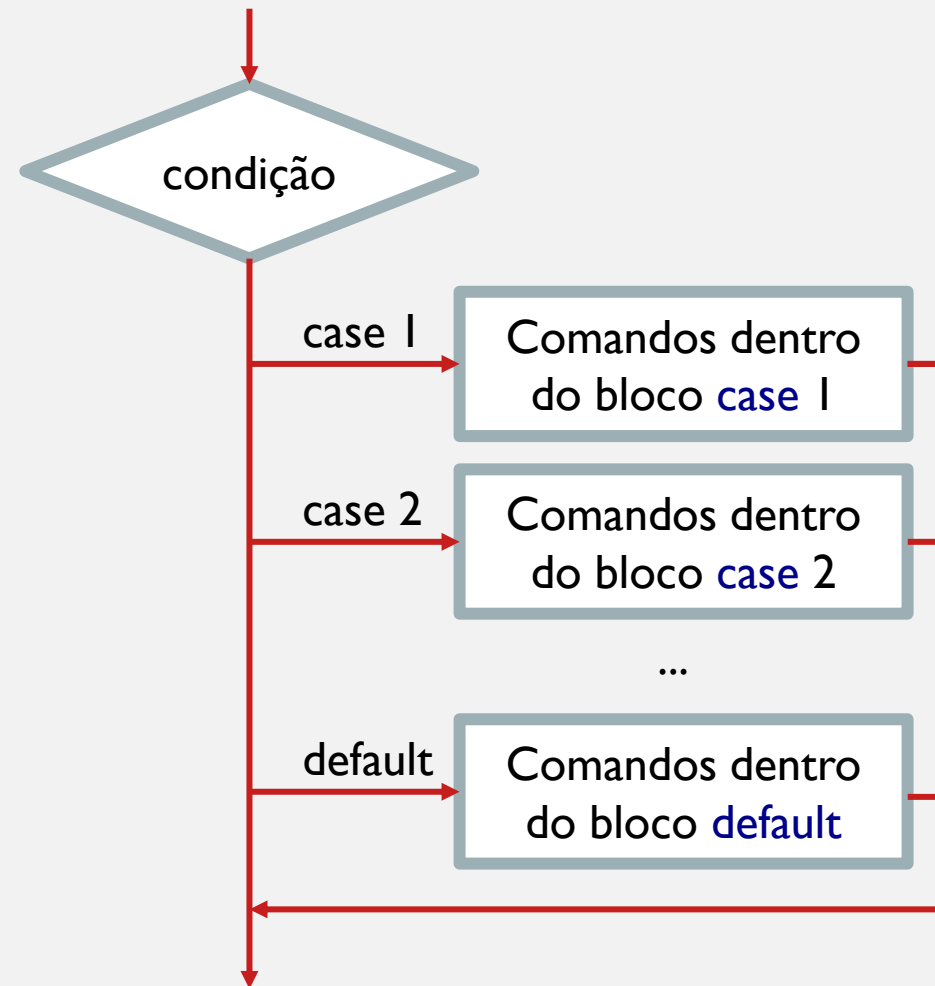
**break;**

**default:**

comandos...

**break;**

}



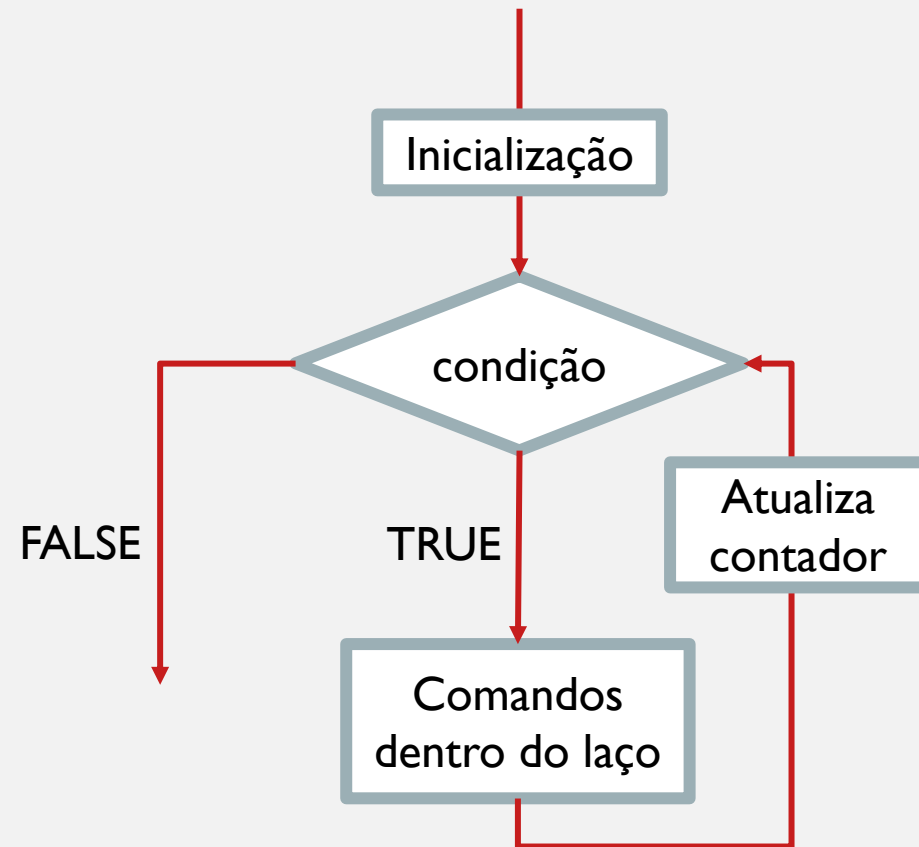
# DECLARAÇÕES DE CONTROLE

- Estruturas de repetição:

- Laço For

```
for (inicialização ; condição ; incremento)
{
    comandos;....
}
```

Pode-se utilizar o comando **break** para encerrar um laço.  
O comando **continue** encerra somente uma iteração

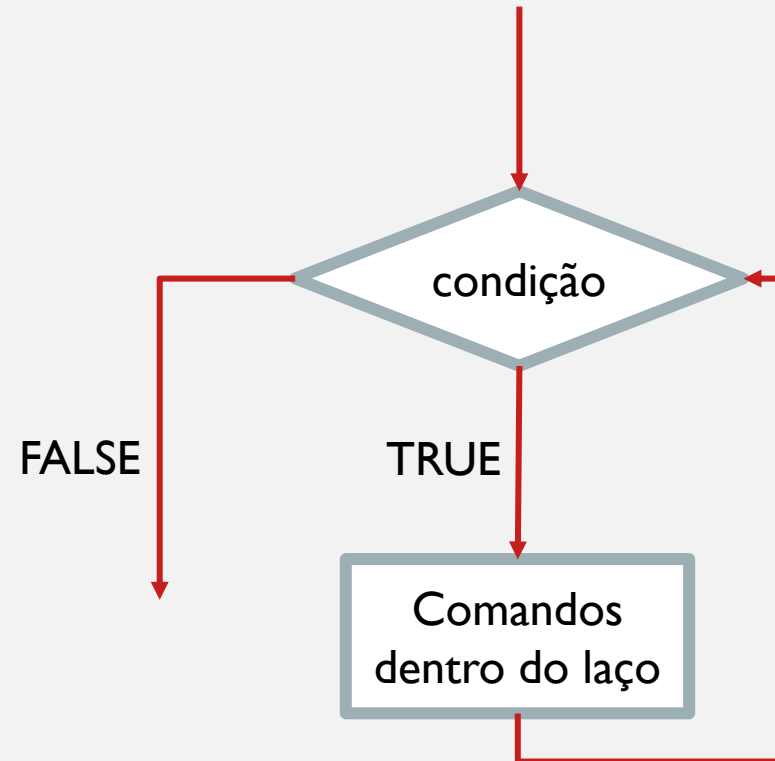


# DECLARAÇÕES DE CONTROLE

- Estruturas de repetição:

- Laço While

```
while (condição)  
{  
    comandos;....  
}
```

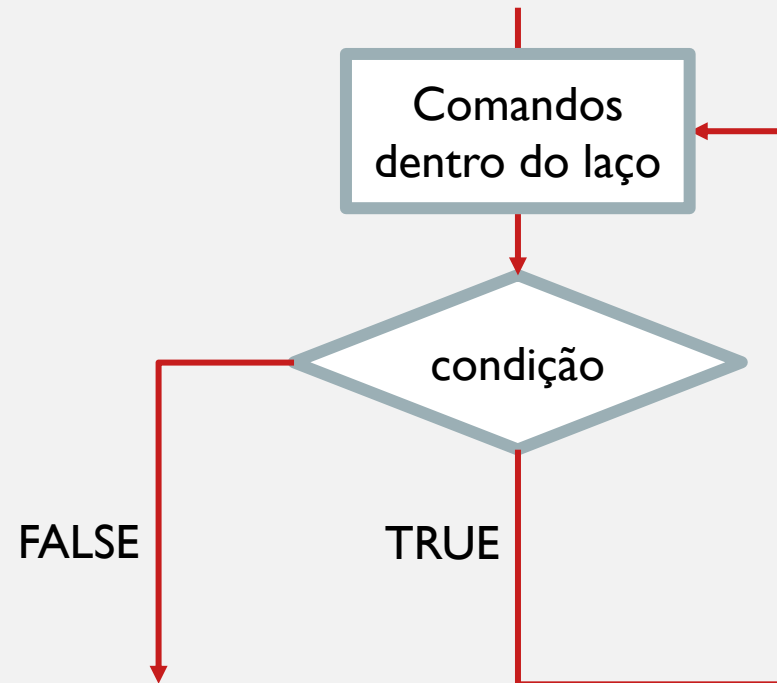


# DECLARAÇÕES DE CONTROLE

- Estruturas de repetição:

- Laço While

```
do  
{  
    comandos;....  
}  
while (condição)
```



# FUNÇÕES

- **Formato geral de uma função em ANSI C:**

```
Tipo_do_retorno  nome_da_função (parâmetros)
{
    comandos...;
}
```

```
Ex: int8_t soma(int8_t a, int8_t b)
{
    return a+b;
}
```

Passagem de parâmetros:

**Por Valor:** passada uma cópia

**Por Referência:** passado o endereço

Declarar o **protótipo** antes de qualquer chamada da função

# ANATOMIA DE UM PROGRAMA

//Comentários contendo informações sobre o programa.  
//Nome, autor, histórico de revisões, etc...

//Arquivos externos

#include <file\_name.h>

//Protótipos das funções

Uma lista das funções que serão utilizadas no programa

//Constantes do programa

#define ON 1

#define OFF 0

//Definição dos handler das interrupções

Utilizadas para relacionar recursos de hardware e software

//Variáveis globais

Lista de variáveis globais

//Programa principal

void main(void)

{

while(1)

{

}

}

//Definição das funções

A implementação, de fato, das funções utilizadas

// Comentário em uma linha

/\* Comentários em múltiplas  
linhas \*/

**Arquivos.h normalmente contém protótipos de  
funções externas ou definições de constantes visíveis  
por todo programa**

#include <arquivos\_do\_compilador.h>

#include "arquivos\_pasta\_local.h"

**Função principal. Início da execução**

**Laço infinito**

## REFERÊNCIAS

### IDE Online

- [https://www.onlinegdb.com/online\\_c\\_compiler#](https://www.onlinegdb.com/online_c_compiler#)

### Material de referência:

- <https://www.learn-c.org/>
- <https://www.tutorialspoint.com/cprogramming>
- <https://www.programiz.com/c-programming>