



# Programação em linguagem C

Rodrigo Reis Gomes

rodrigoreisgomes@gmail.com

# Ementa

- ❑ Introdução ao C
- ❑ Expressões
- ❑ Ponteiros
- ❑ Estruturas de seleção
- ❑ Estruturas de repetição
- ❑ Arranjos
- ❑ Manipulação de textos
- ❑ Tipos de dados definidos pelo programador
- ❑ Arquivos

# Material didático

## ❑ Livro didático

LIVRO E E-BOOK

### Curso de Programação em Linguagem C

**RODRIGO REIS GOMES**

Categoria: Sistema Operacional - Windows

ISBN: 9788539908189

Este livro foi formatado para simplificar o aprendizado de programação em computadores, ao exprimir uma linguagem menos formal em comparação à literatura tradicional e ao explorar a programação com um encadeamento de ensino diferente do habitual. Tradicionalmente, as estruturas de dados e de controle de uma linguagem de programação são ensinadas ao estudante para ele começar a escrever cada um ... [\[leia+\]](#)

[www.lcm.com.br/site/livros/detalhesLivro/f/curso-de-programacao-em-linguagem-c.html](http://www.lcm.com.br/site/livros/detalhesLivro/f/curso-de-programacao-em-linguagem-c.html)

## ❑ Ambiente Virtual de Aprendizado

<https://eadfriburgo.cefet-rj.br/>

---

# Introdução ao C

## ■ Linguagem C

- ❑ Ano de criação: 1972
- ❑ Criador: Dennis Ritchie
- ❑ Objetivo: desenvolver SO Unix
- ❑ Tornou-se LP de propósito geral
- ❑ Há compiladores C para quase todas arquiteturas de computadores

Material: <http://www.codeblocks.org/downloads/26>

---

---

# Introdução ao C

- LP (Linguagem de Programação) tipada
  - Cada estrutura está associada a um tipo

# Introdução ao C

- LP (Linguagem de Programação) tipada
  - Cada estrutura está associada a um tipo
- Tipos da LP

Tipo	Tamanho	Valores
<b>char</b>	<b>08 bits</b>	<b>'a', 'Z', ' ', '+', etc.</b>
short int	16 bits	-32767 a 32767
unsigned short int	16 bits	0 a 65535
<b>int</b> (ou long int)	<b>32 bits</b>	<b>-2147483647 a 2147483647</b>
unsigned long int	32 bits	0 a 4294967295
<b>float</b>	<b>32 bits</b>	<b>Seis dígitos de precisão</b>
double	64 bits	Dez dígitos de precisão
long double	96 bits	Dez dígitos de precisão

# Introdução ao C

- LP (Linguagem de Programação) tipada
  - Cada estrutura está associada a um tipo
- Tipos da LP

Caracteres são representados entre aspas.

Tipo	Tamanho	Valores
<b>char</b>	<b>08 bits</b>	<b>'a', 'Z', ' ', '+', etc.</b>
short int	16 bits	-32767 a 32767
unsigned short int	16 bits	0 a 65535
<b>int</b> (ou long int)	<b>32 bits</b>	<b>-2147483647 a 2147483647</b>
unsigned long int	32 bits	0 a 4294967295
<b>float</b>	<b>32 bits</b>	<b>Seis dígitos de precisão</b>
double	64 bits	Dez dígitos de precisão
long double	96 bits	Dez dígitos de precisão



# Introdução ao C

- LP (Linguagem de Programação) tipada
  - Cada estrutura está associada a um tipo
- Tipos da LP

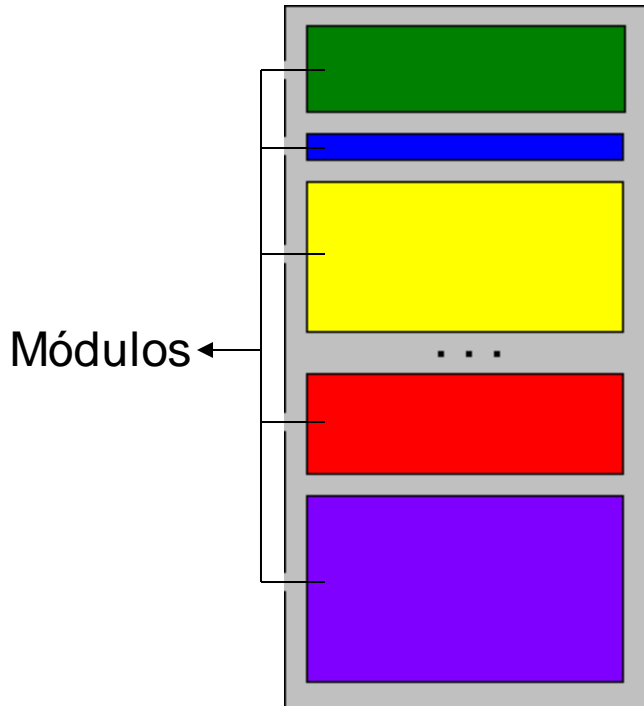
Tipo	Tamanho	Valores
<b>char</b>	<b>08 bits</b>	'a', 'Z', ' ', '+', etc.
short int	16 bits	-32767 a 32767
unsigned short int	16 bits	0 a 65535
<b>int</b> (ou long int)	<b>32 bits</b>	<b>-2147483647 a 2147483647</b>
unsigned long int	32 bits	0 a 4294967295
<b>float</b>	<b>32 bits</b>	<b>Seis dígitos de precisão</b>
double	64 bits	Dez dígitos de precisão
long double	96 bits	Dez dígitos de precisão

Caracteres são representados entre aspas.

Parte inteira e parte fracionária separadas por ponto.

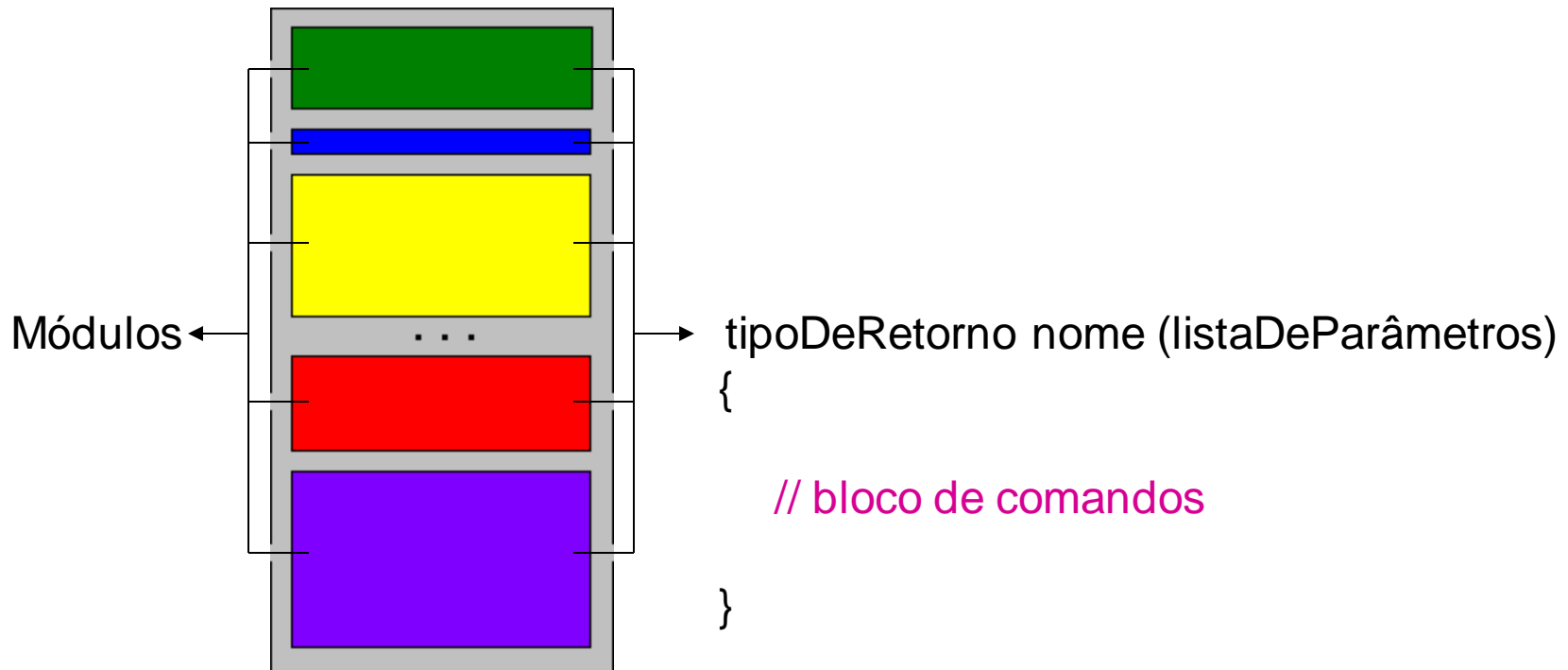
# Introdução ao C

- LP compilada para programas modularizados



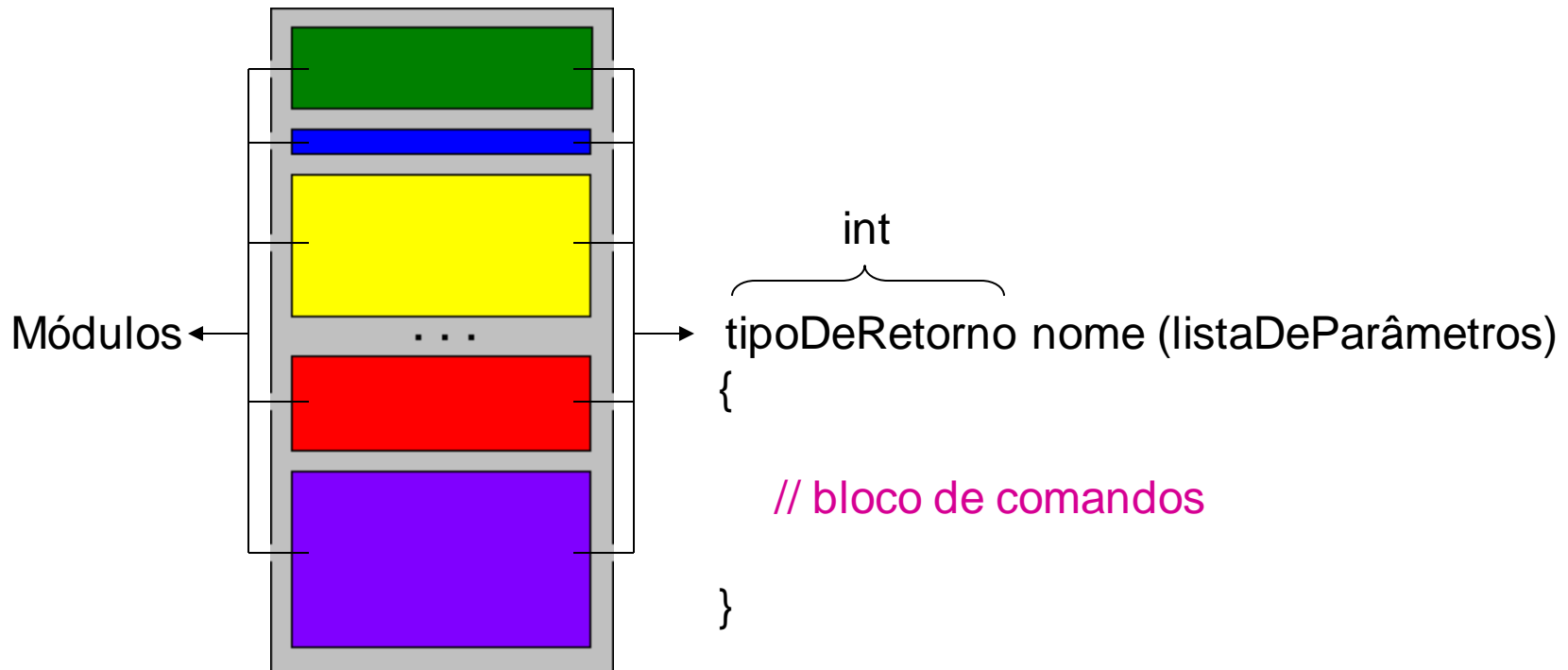
# Introdução ao C

- LP compilada para programas modularizados



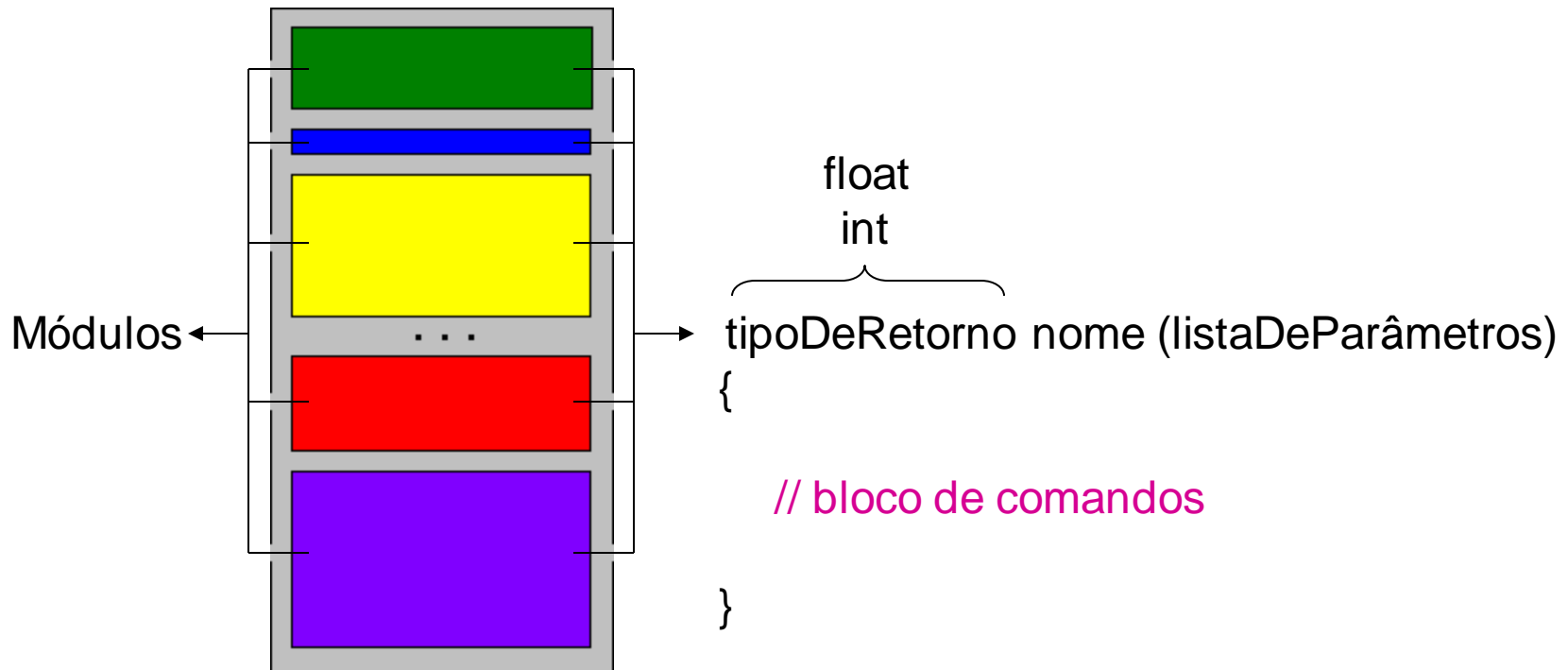
# Introdução ao C

- LP compilada para programas modularizados



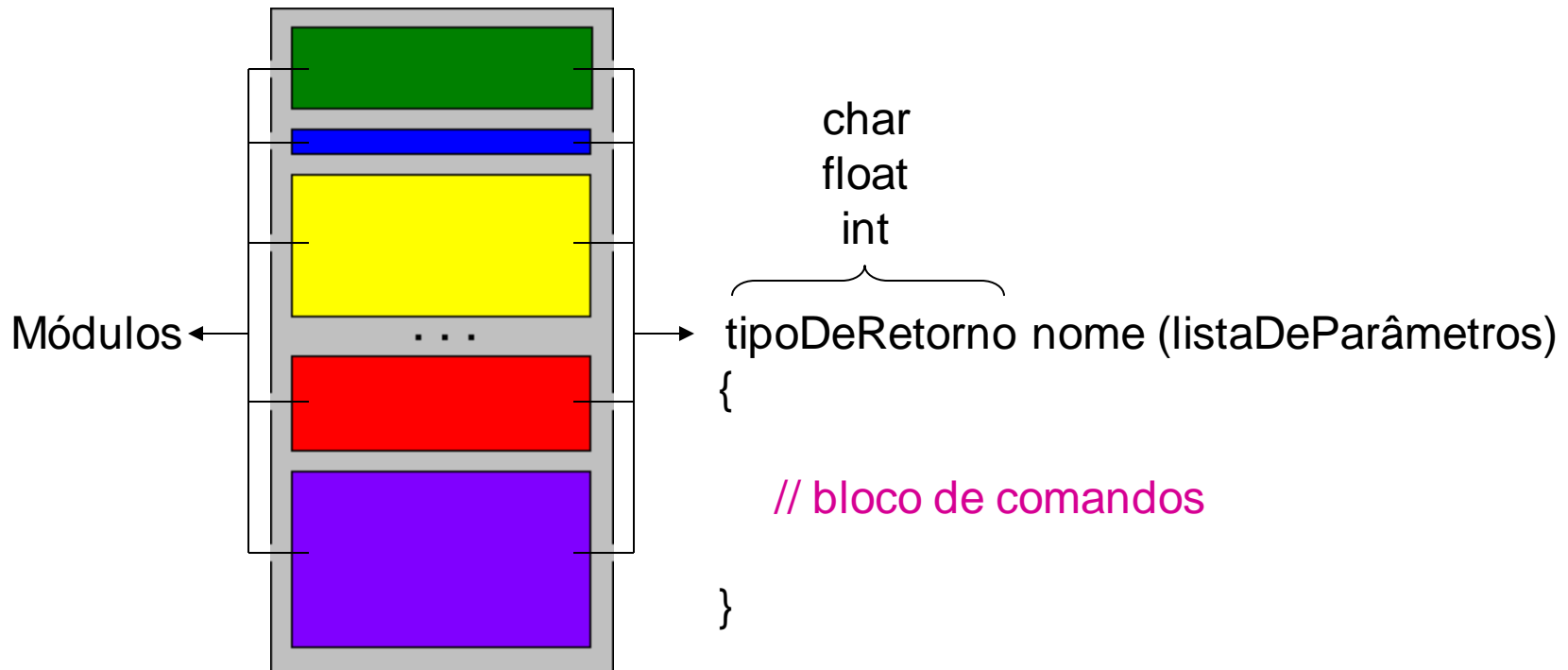
# Introdução ao C

- LP compilada para programas modularizados



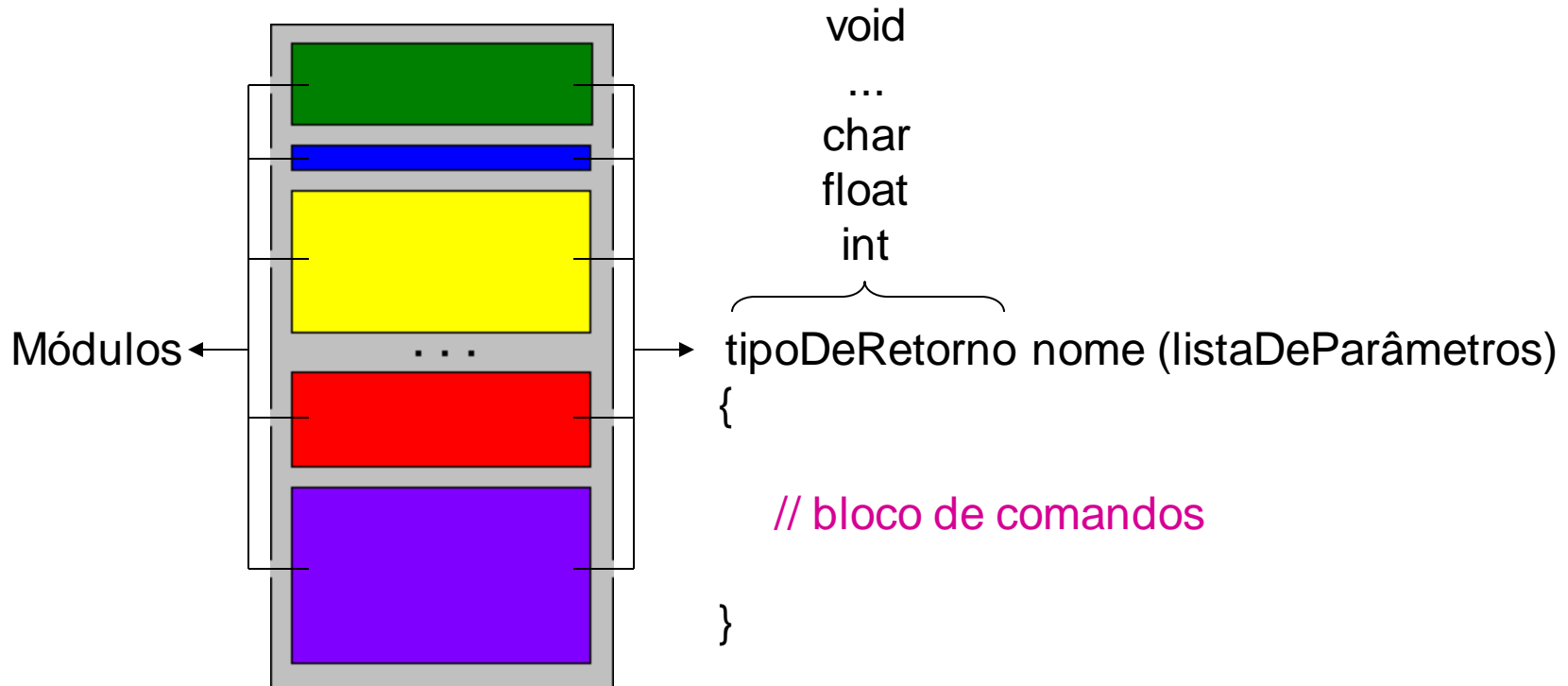
# Introdução ao C

- LP compilada para programas modularizados



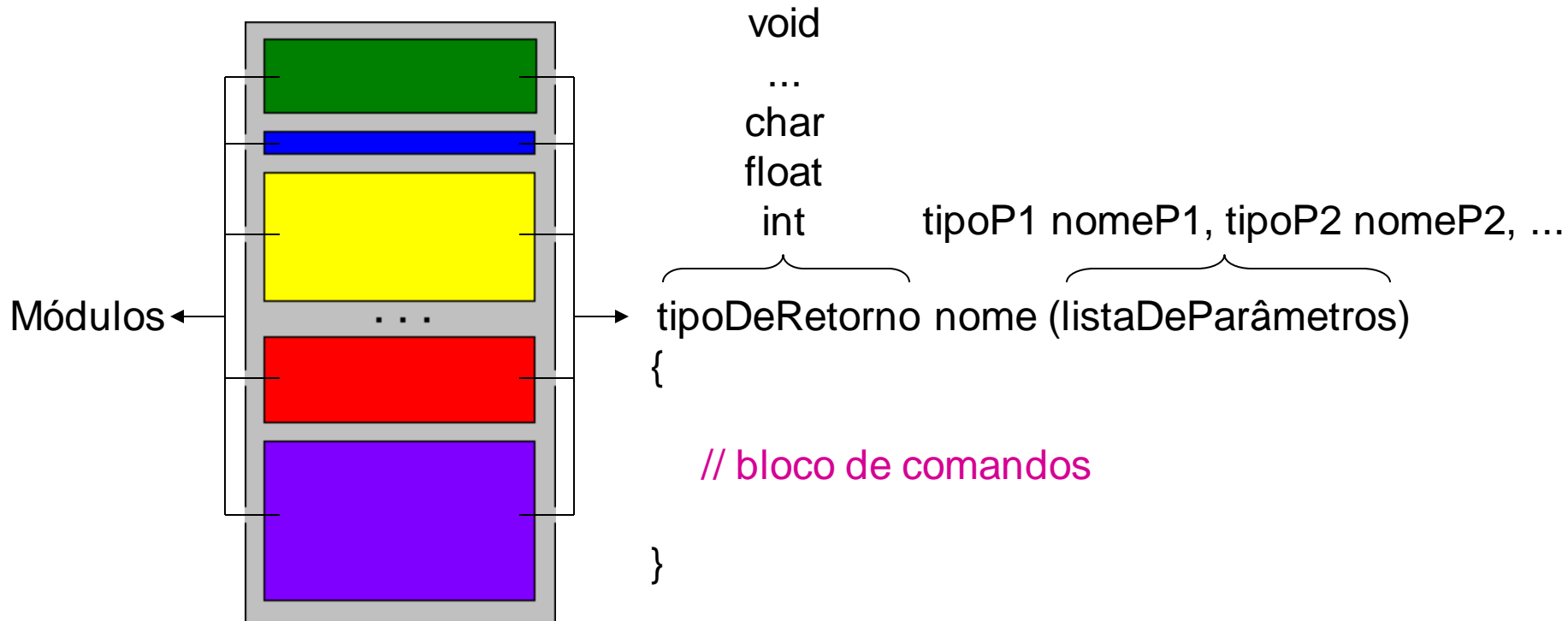
# Introdução ao C

- LP compilada para programas modularizados



# Introdução ao C

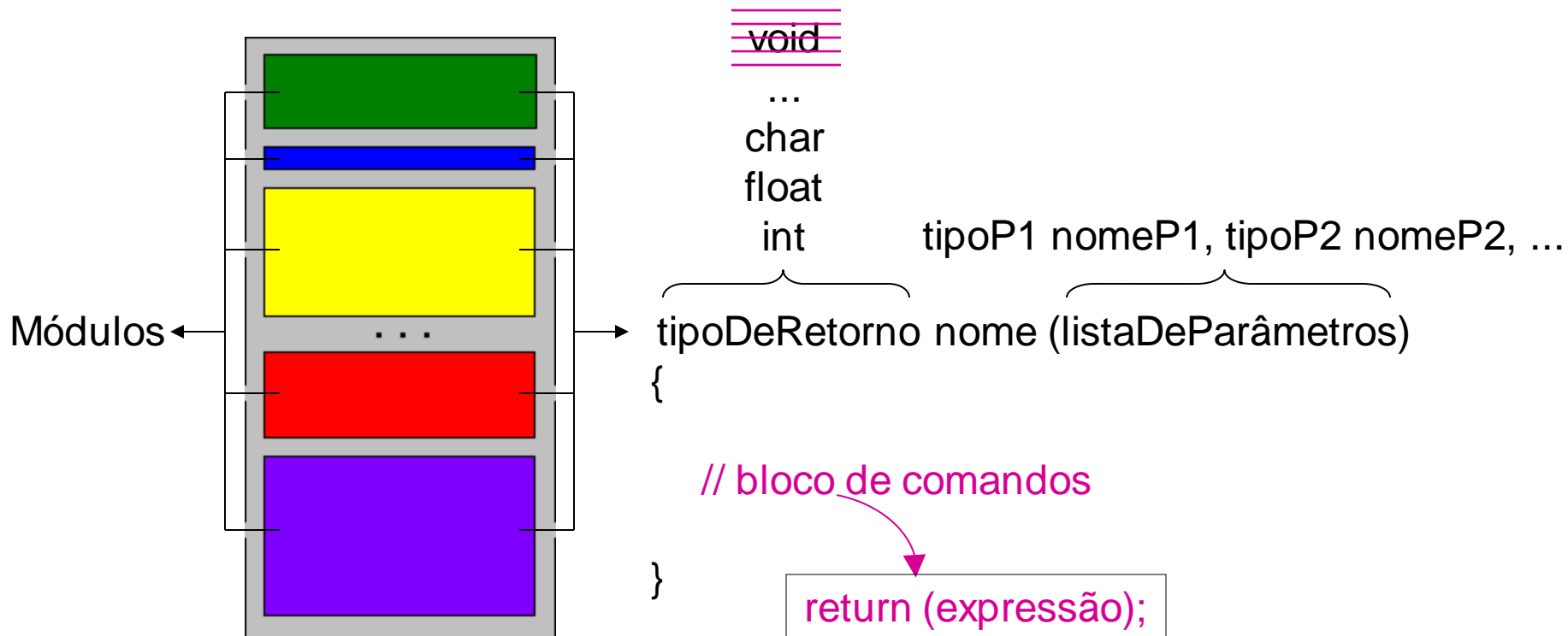
- LP compilada para programas modularizados





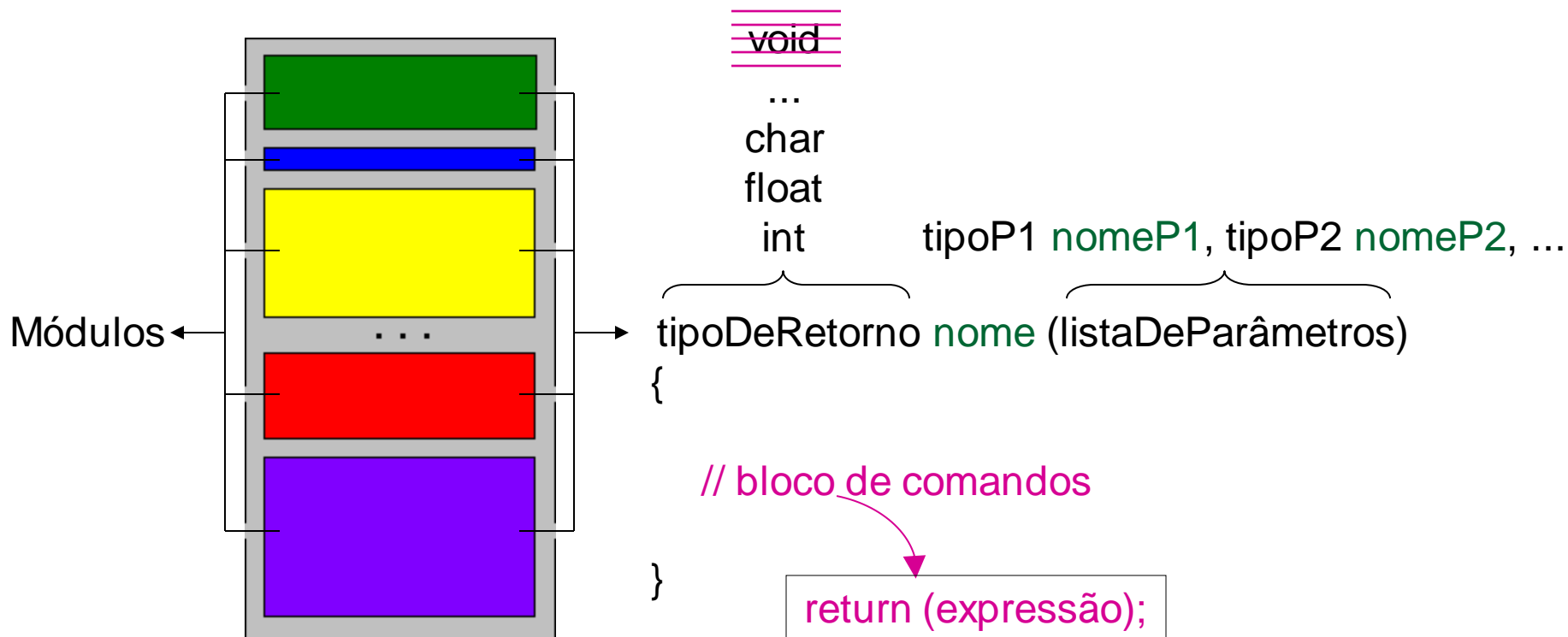
# Introdução ao C

- LP compilada para programas modularizados



# Introdução ao C

- LP compilada para programas modularizados



Os nomes do módulo e dos seus parâmetros devem ser **identificadores**.

# Introdução ao C

## ■ Identificadores

- ❑ Primeiro caractere: letra ou sublinha
- ❑ Demais caracteres: letra ou sublinha ou algarismo
- ❑ Não pode ser uma **palavra reservada**

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

- ❑ Uma letra não pode ter acento nem cedilha

---

# Introdução ao C

- Identificadores

- Exemplos

- A1BC

---

---

# Introdução ao C

- Identificadores

- Exemplos

- A1BC

- XA,1d

---

# Introdução ao C

- Identificadores

- Exemplos

- A1BC

- ~~XA,1d~~

# Introdução ao C

- Identificadores

- Exemplos

- A1BC

- ~~XA,1d~~

- NomeDoAluno

---

# Introdução ao C

- Identificadores

- Exemplos

- A1BC
    - ~~XA,1d~~
    - NomeDoAluno
    - nome\_do\_aluno



---

# Introdução ao C

- Identificadores

- Exemplos

- A1BC
    - ~~XA,1d~~
    - NomeDoAluno
    - nome\_do\_aluno
    - 198\_Aberto

# Introdução ao C

- Identificadores

- Exemplos

- A1BC
    - ~~XA,1d~~
    - NomeDoAluno
    - nome\_do\_aluno
    - ~~198\_Aberto~~

# Introdução ao C

- Identificadores

- Exemplos

- A1BC
    - ~~XA,1d~~
    - NomeDoAluno
    - nome\_do\_aluno
    - ~~198\_Aberto~~
    - float

# Introdução ao C

- Identificadores

- Exemplos

- A1BC
    - ~~XA,1d~~
    - NomeDoAluno
    - nome\_do\_aluno
    - ~~198\_Aberto~~
    - ~~float~~

# Introdução ao C

## ■ Identificadores

### □ Exemplos

- A1BC
- ~~XA,1d~~
- NomeDoAluno
- nome\_do\_aluno
- ~~198\_Aberto~~
- ~~float~~
- média\_final

# Introdução ao C

## ■ Identificadores

### □ Exemplos

- A1BC
- ~~XA,1d~~
- NomeDoAluno
- nome\_do\_aluno
- ~~198\_Aberto~~
- ~~float~~
- ~~média\_final~~

# Introdução ao C

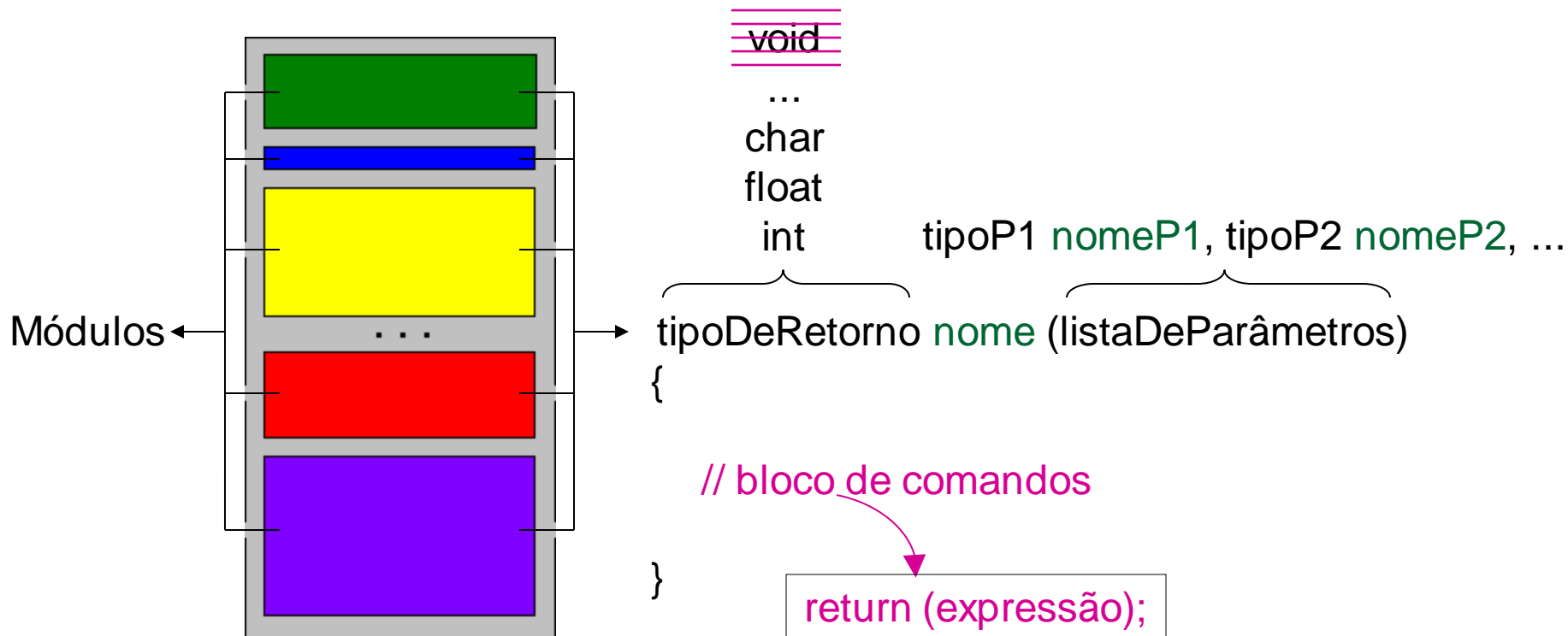
## ■ Identificadores

### □ Exemplos

- A1BC
- ~~XA,1d~~
- NomeDoAluno
- nome\_do\_aluno
- ~~198\_Aberto~~
- ~~float~~
- ~~média\_final~~
- media\_final

# Introdução ao C

- LP compilada para programas modularizados

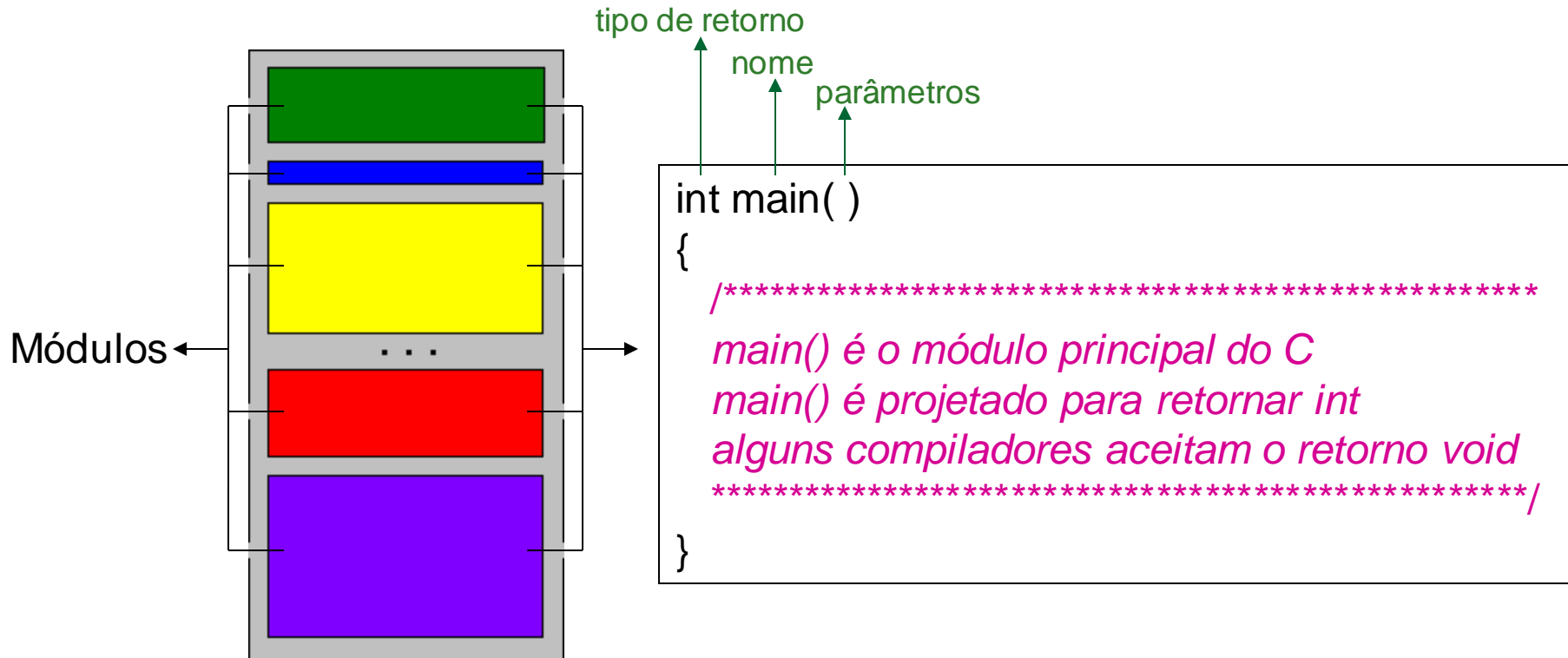


Os nomes do módulo e dos seus parâmetros devem ser **identificadores**.



# Introdução ao C

- LP compilada para programas modularizados



# Introdução ao C

## ■ Saída de dados

### □ *printf*

- Escreve um texto na tela
- Sintaxe

*printf* está programado no arquivo *stdio.h*.

Textos são escritos entre aspas.

`printf(texto a ser escrito);`

## ■ Exemplo

```
#include <stdio.h>
main()
{
    printf("oi"); //chamada ao módulo printf, passando "oi" como argumento
}
```

# Introdução ao C

## ■ Entrada de dados

### □ *getch*

- Lê um caractere do teclado
- Sintaxe

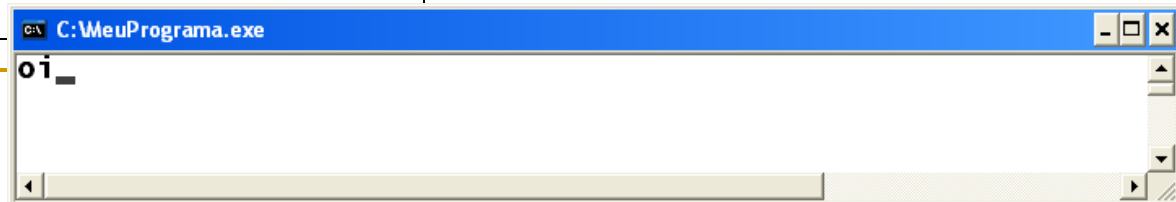
*getch* está programado no arquivo *conio.h*.

`getch();`

*getch* serve para provocar um atraso na execução do programa.

## ■ Exemplo

```
#include <conio.h>
#include <stdio.h>
main()
{
    printf("oi");
    getch();
}
```



# Introdução ao C

## ■ Variáveis

- Posição de memória cujo conteúdo pode ser modificado no decorrer da execução do programa
- Declaração
  - tipo identificador;
  - tipo identificador1, identificador2, ..., identificadorN;
- Exemplos

```
int dia, mes, ano;  
char sexo;  
float salario;
```

# Introdução ao C

## ■ Constantes

- ❑ Posição de memória cujo conteúdo **NÃO** pode ser modificado no decorrer da execução do programa
- ❑ Declaração
  - `const tipo identificador = valor;`
- ❑ Exemplos

```
const float pi = 3.1415;  
const float salarioMinimo = 724.00;
```

# Introdução ao C

## ■ E/S com *printf* e *scanf*

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

```
main()
```

```
{
```

```
    int dia, mes, ano;
```

```
    printf("Digite o dia de seu nascimento: ");
```

```
    scanf("%d", &dia);
```

```
    printf("Digite o mes de seu nascimento: ");
```

```
    scanf("%d", &mes);
```

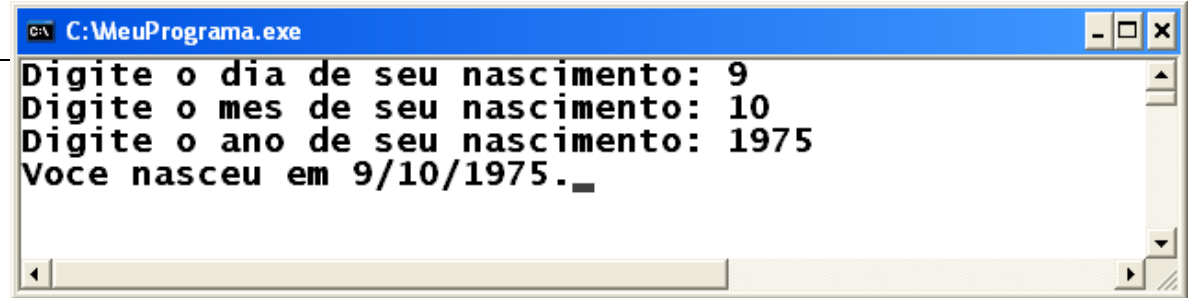
```
    printf("Digite o ano de seu nascimento: ");
```

```
    scanf("%d", &ano);
```

```
    printf("Voce nasceu em %d/%d/%d.", dia, mes, ano);
```

```
    getch();
```

```
}
```



```
C:\MeuPrograma.exe
Digite o dia de seu nascimento: 9
Digite o mes de seu nascimento: 10
Digite o ano de seu nascimento: 1975
Voce nasceu em 9/10/1975._
```

Tipo	Especificador
Caractere (char)	%c
Inteiro (int)	%d
Real (float)	%f
Texto (a ser visto)	%s

`printf(texto, lista de expressões a serem escritas);`  
`scanf(texto, &identificador);`

# Introdução ao C

## ■ Comentários

□ de uma única linha

//comentário

□ de mais de uma linha

/\*

comentário

\*/

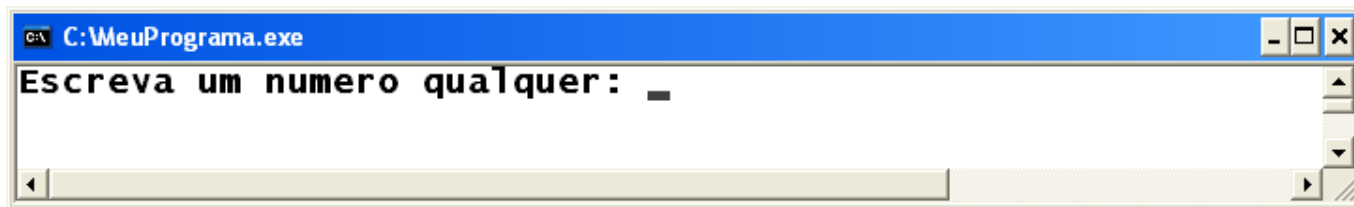
□ Exemplos

```
#include <stdio.h>
#include <conio.h>
main()
{
    int numero;
    printf("Escreva um numero inteiro: ");
    scanf("%d", &numero); // E se for fornecido um número real?
    /*
        Se o usuário inserir um número real (com casas decimais),
        somente a parte inteira deste número será considerada.
    */
    printf("Voce escolheu o numero %d.", numero);
    getch();
}
```

# Introdução ao C

## ■ Formatação de saída numérica

```
#include <stdio.h>
#include <conio.h>
main()
{
    float numero;
    printf("Escreva um numero qualquer: ");
    scanf("%f", &numero);
    // A saída não está formatada.
    printf("Voce escolheu o numero %f.", numero);
    getch();
}
```

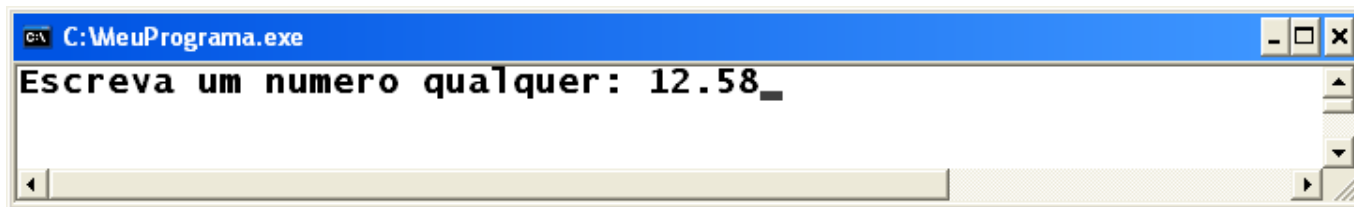




# Introdução ao C

## ■ Formatação de saída numérica

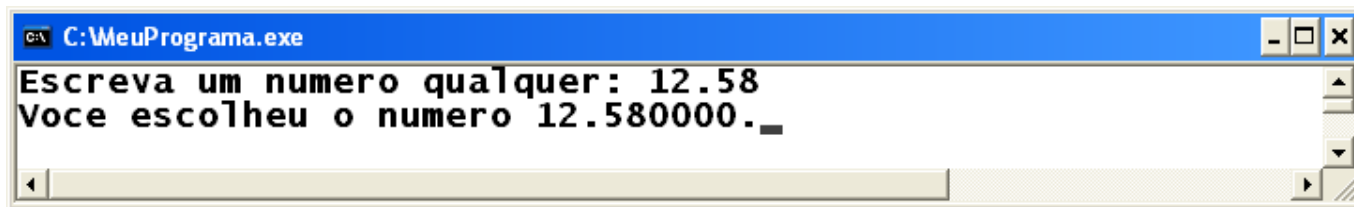
```
#include <stdio.h>
#include <conio.h>
main()
{
    float numero;
    printf("Escreva um numero qualquer: ");
    scanf("%f", &numero);
    // A saída não está formatada.
    printf("Voce escolheu o numero %f.", numero);
    getch();
}
```



# Introdução ao C

## ■ Formatação de saída numérica

```
#include <stdio.h>
#include <conio.h>
main()
{
    float numero;
    printf("Escreva um numero qualquer: ");
    scanf("%f", &numero);
    // A saída não está formatada.
    printf("Voce escolheu o numero %f.", numero);
    getch();
}
```



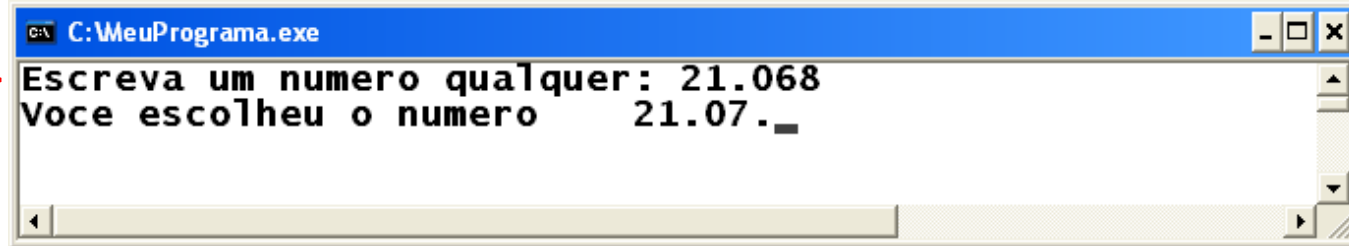
```
C:\MeuPrograma.exe
Escreva um numero qualquer: 12.58
Voce escolheu o numero 12.580000. _
```

# Introdução ao C

## ■ Formatação de saída numérica

```
#include <stdio.h>
#include <conio.h>
main()
```

```
{
    float numero;
    printf("Escreva um numero qualquer: ");
    scanf("%f", &numero);
    printf("Voce escolheu o numero %8.2f.", numero);
    getch();
}
```



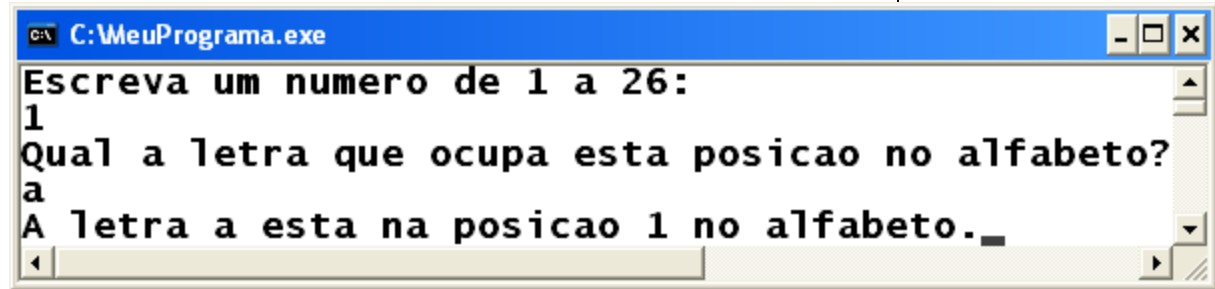
```
C:\MeuPrograma.exe
Escreva um numero qualquer: 21.068
Voce escolheu o numero 21.07._
```

**%*tamanhoMínimoNoNúmero.quantidadeDeCasasDecimais*f**

# Introdução ao C

## ■ Caracteres de barra invertida

```
#include <stdio.h>
#include <conio.h>
main()
{
    int posicao;
    char letra;
    printf("Escreva um número de 1 a 26:\n");
    scanf("%d", &posicao);
    printf("Qual a letra que ocupa esta posição no alfabeto?\n");
    scanf("\n%c", &letra);
    printf("A letra %c esta na posicao %d no alfabeto.", letra, posicao);
    getch();
}
```

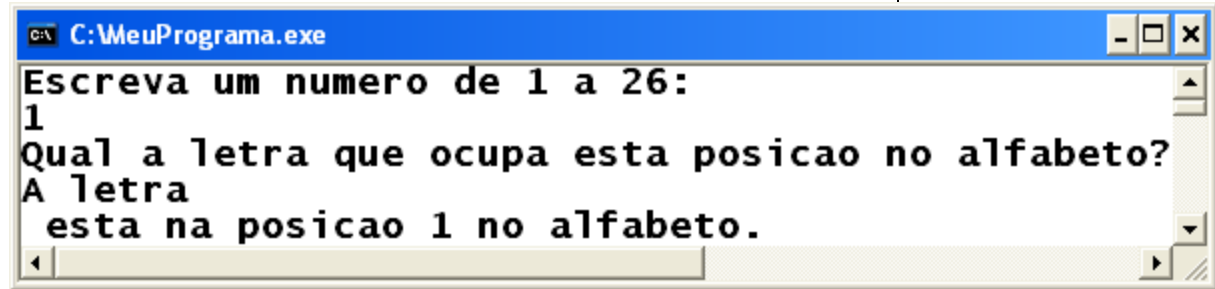


```
C:\MeuPrograma.exe
Escreva um número de 1 a 26:
1
Qual a letra que ocupa esta posicao no alfabeto?
a
A letra a esta na posicao 1 no alfabeto.
```

# Introdução ao C

## ■ Caracteres de barra invertida

```
#include <stdio.h>
#include <conio.h>
main()
{
    int posicao;
    char letra;
    printf("Escreva um número de 1 a 26:\n");
    scanf("%d", &posicao);
    printf("Qual a letra que ocupa esta posição no alfabeto?\n");
    scanf("%c", &letra);
    printf("A letra %c esta na posicao %d no alfabeto.", letra, posicao);
    getch();
}
```



# Introdução ao C

## ■ Caracteres de barra invertida

Caractere	Significado
\b	Retrocesso (BS)
\f	Alimentação de formulário (FF)
\r	Retorno de carro (CR)
\t	Tabulação horizontal (HT)
\"	Aspas
\'	Apóstrofo
\0	Nulo
\\	Barra invertida
\v	Tabulação vertical
\a	Alerta ( <i>beep</i> )

# Introdução ao C

- Textos como tipos de dados
  - Declaração da variável

```
char * identificador;
```

- Antes de usar a variável

```
identificador = (char *)malloc(sizeof(char));
```

- Depois de usar a variável

```
free(identificador);
```

*malloc e free programadas  
no arquivo `stdlib.h`.*

# Introdução ao C

## ■ Textos como tipos de dados

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

```
main()
{
    char * nome;
    nome = (char *)malloc(sizeof(char));
    printf("Digite seu nome: ");
    scanf("%s", nome);
```

```
/*
```

Atenção: para lermos variáveis de texto, não incluímos '&' antes do identificador no comando "scanf".

Inconveniente: O comando "scanf" entende o caractere espaço como o final de um valor.

```
*/
```

```
printf("Bem vindo ao mundo da computacao, %s.", nome);
free(nome);
getch();
}
```



# Introdução ao C

## ■ Textos como tipos de dados

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

```
main()
```

```
{
```

```
    char * nome;
```

```
    nome = (char *)malloc(sizeof(char));
```

```
    printf("Digite seu nome: ");
```

```
    scanf("%s", nome);
```

```
    /*
```

Atenção: para lermos variáveis de texto, não incluímos '&' antes do identificador no comando "scanf".

Inconveniente: O comando "scanf" entende o caractere espaço como o final de um valor.

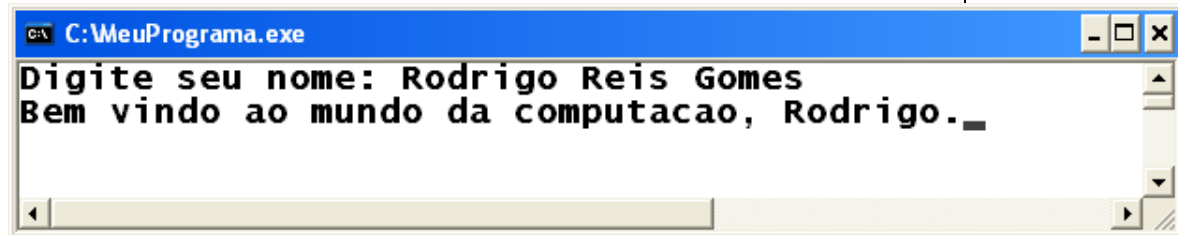
```
    */
```

```
    printf("Bem vindo ao mundo da computacao, %s.", nome);
```

```
    free(nome);
```

```
    getch();
```

```
}
```

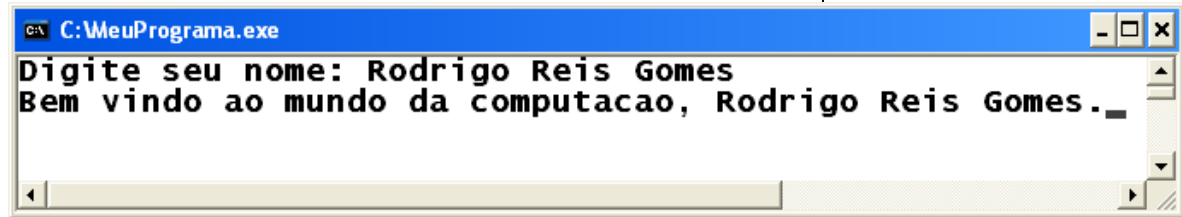


# Introdução ao C

## ■ Textos como tipos de dados

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

```
main()
{
    char * nome;
    nome = (char *)malloc(sizeof(char));
    printf("Digite seu nome: ");
    gets(nome);
    printf("Bem vindo ao mundo da computacao, %s.", nome);
    free(nome);
    getch();
}
```



- *gets* (leitura de um texto): `gets(identificador);`

---

# Introdução ao C

- Exercícios do capítulo 1



---

# Expressões


- Combinação de operadores que atuam sobre constantes, variáveis e funções

# Expressões

- Combinação de operadores que atuam sobre constantes, variáveis e funções
- Operadores:
  - atribuição e *cast*
  - aritméticos
  - relacionais
  - lógicos
  - de posição de memória (próximo capítulo)
  - de *bit a bit* (apêndice)


# Expressões

## ■ Atribuição

  
identificador = expressão;

# Expressões

## ■ Atribuição

  
identificador = expressão;

## □ Exemplos

A = 2;

Estado = "RJ";

NOTA = 10;

X = 2.5;


SEXO = 'M';

B = A;



# Expressões

## ■ Atribuição

  
identificador = expressão;

### □ Exemplos

A = 2;	NOTA = 10;	SEXO = 'M';
Estado = "RJ";	X = 2.5;	B = A;

### □ Identificador e expressões com mesmo tipo

- inteiro X caractere
  - inteiro X real
- } **exceções**

---

# Expressões

- Atribuição

- inteiro = caractere;

# Expressões

- Atribuição

- inteiro = caractere;

- O identificador recebe o código ASCII do caractere

# Expressões

## ■ Atribuição

- inteiro = caractere;

  - O identificador recebe o código ASCII do caractere

- caractere = inteiro;

# Expressões

## ■ Atribuição

- inteiro = caractere;

- O identificador recebe o código ASCII do caractere

- caractere = inteiro;

- O identificador recebe o caractere associado ao código ASCII do inteiro módulo 256

# Expressões

## ■ Atribuição

- inteiro = caractere;

  - O identificador recebe o código ASCII do caractere

- caractere = inteiro;

  - O identificador recebe o caractere associado ao código ASCII do inteiro módulo 256

- inteiro = real;

# Expressões

## ■ Atribuição

- inteiro = caractere;

- O identificador recebe o código ASCII do caractere

- caractere = inteiro;

- O identificador recebe o caractere associado ao código ASCII do inteiro módulo 256

- inteiro = real;

- O identificador recebe um valor truncado

# Expressões

## ■ Atribuição

- inteiro = caractere;

- O identificador recebe o código ASCII do caractere

- caractere = inteiro;

- O identificador recebe o caractere associado ao código ASCII do inteiro módulo 256

- inteiro = real;

- O identificador recebe um valor truncado

- real = inteiro;



# Expressões

## ■ Atribuição

- inteiro = caractere;

- O identificador recebe o código ASCII do caractere

- caractere = inteiro;

- O identificador recebe o caractere associado ao código ASCII do inteiro módulo 256

- inteiro = real;

- O identificador recebe um valor truncado

- real = inteiro;

- O identificador recebe um valor com a parte fracionária igual a zero

# Expressões

## ■ *Casts*

(tipo) expressão

### □ Exemplo

```
int x;
```

```
x = 5;
```

```
printf("%f\n", (float)x/2);
```

# Expressões

## ■ *Casts*

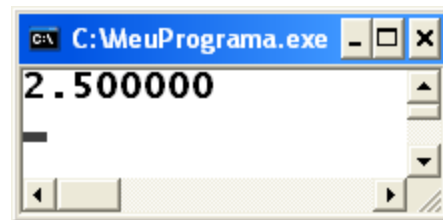
(tipo) expressão

### □ Exemplo

```
int x;
```

```
x = 5;
```

```
printf("%f\n", (float)x/2);
```



# Expressões

## ■ *Casts*

(tipo) expressão

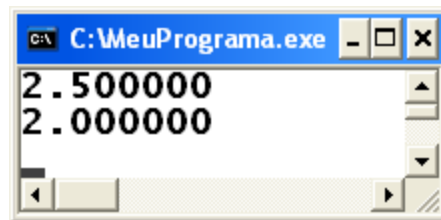
### □ Exemplo

```
int x;
```

```
x = 5;
```

```
printf("%f\n", (float)x/2);
```

```
printf("%f\n", (float)(x/2));
```



# Expressões

## ■ Operadores aritméticos

Operador	Operação	Tipo do operador	Tipo do(s) operando(s)	Forma do operador	Resultado
+	Adição	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
-	Subtração	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
*	Multiplicação	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
/	Divisão	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
%	Resto da divisão	Binário	Inteiro	Identificador ou valor	Inteiro
--	Decremento	Unário	Inteiro	Identificador	Inteiro
++	Incremento	Unário	Inteiro	Identificador	Inteiro
-	Menos unário	Unário	Inteiro ou real	Identificador	Inteiro ou real

# Expressões

## ■ Operadores aritméticos

Operador	Operação	Tipo do operador	Tipo do(s) operando(s)	Forma do operador	Resultado
+	Adição	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
-	Subtração	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
*	Multiplicação	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
/	Divisão	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
%	Resto da divisão	Binário	Inteiro	Identificador ou valor	Inteiro
--	Decremento	Unário	Inteiro	Identificador	Inteiro
++	Incremento	Unário	Inteiro	Identificador	Inteiro
-	Menos unário	Unário	Inteiro ou real	Identificador	Inteiro ou real

Expressão	Resultado
1 + 2	3
5.0 - 1	4.0
2 * 1.5	3.0
5 / 2	2
5 / 2.0	2.5
5 % 2	1

# Expressões

## ■ Operadores aritméticos

Operador	Operação	Tipo do operador	Tipo do(s) operando(s)	Forma do operador	Resultado
+	Adição	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
-	Subtração	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
*	Multiplicação	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
/	Divisão	Binário	Inteiro ou real	Identificador ou valor	Inteiro ou real
%	Resto da divisão	Binário	Inteiro	Identificador ou valor	Inteiro
--	Decremento	Unário	Inteiro	Identificador	Inteiro
++	Incremento	Unário	Inteiro	Identificador	Inteiro
-	Menos unário	Unário	Inteiro ou real	Identificador	Inteiro ou real

Expressão	Resultado	Expressão	Resultado
1 + 2	3	x + 2	4
5.0 - 1	4.0	x - y	-3.0
2 * 1.5	3.0	y * 2	10.0
5 / 2	2	-x	-2
5 / 2.0	2.5	++x	Modifica o valor de x para 3
5 % 2	1	--x	Modifica o valor de x para 1

x = 2;  
y = 5.0;

# Expressões

## ■ Um exemplo usando módulos

$$s = s_0 + v_0 t + 0,5at^2$$

$$s - s_0 = v_0 t + 0,5at^2$$

$$\text{deslocamento} = v_0 t + 0,5at^2$$

```
#include <stdio.h>
#include <conio.h>
float distancia (float v, float t, float a)
{
    float r;
    r = v*t + 0.5*a*t*t;
    return r;
}
main()
{
    float vi, ac, it, dp;
    printf("FORNECA A VELOCIDADE INICIAL: ");
    scanf("%f", &vi);
    printf("FORNECA A ACELERACAO: ");
    scanf("%f", &ac);
    printf("FORNECA O INTERVALO DE TEMPO: ");
    scanf("%f", &it);
    dp = distancia(vi, it, ac);
    printf("DISTANCIA PERCORRIDA: %f", dp);
    getch();
}
```



# Expressões

## ■ Funções numéricas predefinidas

Função	Finalidade	Tipo do(s) argumento(s)	Tipo do resultado
<code>fabs(x)</code>	Valor absoluto	Real	Real
<code>floor(x)</code>	Arredondamento para baixo	Real	Real
<code>ceil(x)</code>	Arredondamento para cima	Real	Real
<code>sqrt(x)</code>	Raiz quadrada	Real	Real
<code>pow(x, y)</code>	Potenciação ( $x^y$ )	Real, Real	Real
<code>exp(x)</code>	Exponencial ( $e^x$ )	Real	Real
<code>log(x)</code>	Logaritmo natural	Real	Real
<code>log10(x)</code>	Logaritmo na base 10	Real	Real

Expressão	Resultado
<code>fabs(-2.5)</code>	2.5
<code>fabs(8.0)</code>	8.0
<code>floor(5.234)</code>	5.0
<code>ceil(2.78)</code>	3.0
<code>sqrt(9.0)</code>	3.0
<code>pow(2.0,3.0)</code>	8.0
<code>pow(3.0,2.0)</code>	9.0
<code>exp(1.0)</code>	2.718282
<code>log(2.718282)</code>	1.0
<code>log10(10.0)</code>	1.0

É preciso incluir o arquivo de cabeçalho *math.h*

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

# Expressões

## ■ Operadores relacionais

Operador	Operação
<code>==</code>	Igual
<code>&gt;</code>	Maior
<code>&lt;</code>	Menor
<code>&gt;=</code>	Maior ou igual
<code>&lt;=</code>	Menor ou igual
<code>!=</code>	Diferente

Expressão	Resultado
<code>1 == 2</code>	0
<code>2.0 == 2</code>	1
<code>5 &gt; 2</code>	1
<code>3 &lt;= 3</code>	1
<code>3.4 != 5.7</code>	1
<code>2 + 3 != 5</code>	0
<code>'A' == 'a'</code>	0

- ❑ Binários (dois operandos)
- ❑ Operandos
  - caracteres e números
- ❑ Resultado
  - *booleano* (0 ou 1)

# Expressões

- Operadores relacionais e o tipo char
  - Tabela ASCII (American Standard Code for Information Interchange)

Caractere	Código ASCII
'a'	97
'b'	98
'c'	99
...	
'y'	121
'z'	122

Caractere	Código ASCII
'A'	65
'B'	66
'C'	67
...	
'Y'	89
'Z'	90

Caractere	Código ASCII
'0'	48
'1'	49
'2'	50
...	
'8'	56
'9'	57

# Expressões

- Operadores relacionais e o tipo char \*
- Comparação caractere por caractere

“JOAO” < “JOSE”

(ASCII('J') = 74) = (ASCII('J') = 74)

(ASCII('O') = 79) = (ASCII('O') = 79)

(ASCII('A') = 65) < (ASCII('S') = 83)

# Expressões

## ■ Operadores lógicos

Operador	Operação
!	não (negação)
&&	e (conjunção)
	ou (disjunção)

Expressão	Resultado
(1 > 2) && (3 > 2)	0
(1 > 2)    (3 > 2)	1
! (1 > 2)	1

- ❑ Binários (&& e ||)
- ❑ Unário (!)
- ❑ Operandos (*booleano*)
- ❑ Resultado (*booleano*)
  - && retorna 1 quando ambos os operandos forem 1
  - || retorna 0 quando ambos os operandos forem 0
  - ! inverte o valor lógico do seu operando

# Expressões

## ■ Prioridade dos operadores

Prioridade	Operadores
1ª	Parênteses mais internos
2ª	Funções
3ª	++ -- -(menos unário)
4ª	* / %
5ª	+ -
6ª	> < >= <=
7ª	= !=
8ª	!
9ª	&&
10ª	
11ª	=
12ª	Operador mais à esquerda na expressão

---

# Expressões

- Exercícios do capítulo 2





# Ponteiros

- Variável cujo conteúdo é uma posição de memória
- Declaração

tipo \* identificador;

- Operadores de posição de memória

Operador	Significado
&	O endereço de uma posição de memória.
*	O conteúdo de uma posição de memória.

# Ponteiros

## ■ Exemplo

```
01 main()
02 {
03     int *p;
04     int i;
05     i = 5;
06     *p = i;
07     printf("i = %d\n", i);
08     printf("&i = %d\n", &i);
09     printf("p = %d\n", p);
10     printf("&p = %d\n", &p);
11     printf("*p = %d\n", *p);
12 }
```

identificador	endereço	conteúdo
	2147319808	0
p	2359156	2147319808

# Ponteiros

## ■ Exemplo

```
01 main()
02 {
03     int *p;
04     int i;
05     i = 5;
06     *p = i;
07     printf("i = %d\n", i);
08     printf("&i = %d\n", &i);
09     printf("p = %d\n", p);
10     printf("&p = %d\n", &p);
11     printf("*p = %d\n", *p);
12 }
```

identificador	endereço	conteúdo
	2147319808	0
p	2359156	2147319808
i	2359152	4199328

# Ponteiros

## ■ Exemplo

```
01 main()
02 {
03     int *p;
04     int i;
05     i = 5;
06     *p = i;
07     printf("i = %d\n", i);
08     printf("&i = %d\n", &i);
09     printf("p = %d\n", p);
10     printf("&p = %d\n", &p);
11     printf("*p = %d\n", *p);
12 }
```

identificador	endereço	conteúdo
	2147319808	0
p	2359156	2147319808
i	2359152	5

# Ponteiros

## ■ Exemplo

```
01 main()
02 {
03     int *p;
04     int i;
05     i = 5;
06     *p = i;
07     printf("i = %d\n", i);
08     printf("&i = %d\n", &i);
09     printf("p = %d\n", p);
10     printf("&p = %d\n", &p);
11     printf("*p = %d\n", *p);
12 }
```

identificador	endereço	conteúdo
	2147319808	5
p	2359156	2147319808
i	2359152	5

# Ponteiros

## ■ Exemplo

```
01 main()
02 {
03   int *p;
04   int i;
05   i = 5;
06   *p = i;
07   printf("i = %d\n", i);
08   printf("&i = %d\n", &i);
09   printf("p = %d\n", p);
10   printf("&p = %d\n", &p);
11   printf("*p = %d\n", *p);
12 }
```

identificador	endereço	conteúdo
	2147319808	5
p	2359156	2147319808
i	2359152	5

```
i = 5
&i = 2359152
p = 2147319808
&p = 2359156
*p = 5
```

# Ponteiros

## ■ Outro exemplo

```
01 main()
02 {
03     int *p;
04     int i;
05     i = 5;
06     p = &i;
07     printf("i = %d\n", i);
08     printf("&i = %d\n", &i);
09     printf("p = %d\n", p);
10     printf("&p = %d\n", &p);
11     printf("*p = %d\n", *p);
12 }
```

identificador	endereço	conteúdo
p	2359156	2147340288

# Ponteiros

## ■ Outro exemplo

```
01 main()
02 {
03     int *p;
04     int i;
05     i = 5;
06     p = &i;
07     printf("i = %d\n", i);
08     printf("&i = %d\n", &i);
09     printf("p = %d\n", p);
10     printf("&p = %d\n", &p);
11     printf("*p = %d\n", *p);
12 }
```

identificador	endereço	conteúdo
p	2359156	2147340288
i	2359152	4199368



# Ponteiros

## ■ Outro exemplo

```
01 main()
02 {
03     int *p;
04     int i;
05     i = 5;
06     p = &i;
07     printf("i = %d\n", i);
08     printf("&i = %d\n", &i);
09     printf("p = %d\n", p);
10     printf("&p = %d\n", &p);
11     printf("*p = %d\n", *p);
12 }
```

identificador	endereço	conteúdo
p	2359156	2147340288
i	2359152	5

# Ponteiros

## ■ Outro exemplo

```
01 main()
02 {
03     int *p;
04     int i;
05     i = 5;
06     p = &i;
07     printf("i = %d\n", i);
08     printf("&i = %d\n", &i);
09     printf("p = %d\n", p);
10     printf("&p = %d\n", &p);
11     printf("*p = %d\n", *p);
12 }
```

identificador	endereço	conteúdo
p	2359156	2359152
i	2359152	5

# Ponteiros

## ■ Outro exemplo

```
01 main()
02 {
03     int *p;
04     int i;
05     i = 5;
06     p = &i;
07     printf("i = %d\n", i);
08     printf("&i = %d\n", &i);
09     printf("p = %d\n", p);
10     printf("&p = %d\n", &p);
11     printf("*p = %d\n", *p);
12 }
```

identificador	endereço	conteúdo
p	2359156	2359152
i	2359152	5

```
i = 5
&i = 2359152
p = 2359152
&p = 2359156
*p = 5
```

# Ponteiros

## ■ Mais um exemplo

```
01 main()
02 {
03   int x, y;
04   int *px = &x, *py = &y;
05   printf("px= %d\n", px);
06   printf("py= %d\n", py);
07   printf("px-py= %d\n", px-py);
08   printf("py-px= %d\n", py-px);
09 }
```

identificador	endereço	conteúdo
x	2359156	2147319808
y	2359152	4199376

# Ponteiros

## ■ Mais um exemplo

```
01 main()
02 {
03   int x, y;
04   int *px = &x, *py = &y;
05   printf("px= %d\n", px);
06   printf("py= %d\n", py);
07   printf("px-py= %d\n", px-py);
08   printf("py-px= %d\n", py-px);
09 }
```

identificador	endereço	conteúdo
x	2359156	2147319808
y	2359152	4199376
px	2359148	2359156
py	2359144	2359152

# Ponteiros

## ■ Mais um exemplo

```
01 main()
02 {
03   int x, y;
04   int *px = &x, *py = &y;
05   printf("px= %d\n", px);
06   printf("py= %d\n", py);
07   printf("px-py= %d\n", px-py);
08   printf("py-px= %d\n", py-px);
09 }
```

identificador	endereço	conteúdo
x	2359156	2147319808
y	2359152	4199376
px	2359148	2359156
py	2359144	2359152

```
px = 2359156
py = 2359152
px-py = 1
py-px = -1
```

# Ponteiros

## ■ Mais um exemplo

```
01 main()
02 {
03   int x, y;
04   int *px = &x, *py = &y;
05   printf("px= %d\n", px);
06   printf("py= %d\n", py);
07   printf("px-py= %d\n", px-py);
08   printf("py-px= %d\n", py-px);
09 }
```

identificador	endereço	conteúdo
x	2359156	2147319808
y	2359152	4199376
px	2359148	2359156
py	2359144	2359152

O tipo *int* representa valores de 4 *bytes*.

```
px = 2359156
py = 2359152
px-py = 1
py-px = -1
```

# Ponteiros

- Alocação e liberação dinâmica de memória
  - Quando o usuário determina o valor a ser apontado por um ponteiro

```
identificador = (ponteiroParaTipo)malloc(quantidadeDeMemoriaEmBytes);
```

*malloc e free pertencem a  
stdlib.h.*

·  
·  
·

Antes de usar a variável.

```
free(identificador);
```

Depois de ter usado a variável.



# Ponteiros

## ■ Passagem de argumentos por valor

```
01 #include <stdio.h>
02 #include <conio.h>

03 void troca(int x, int y)
04 {
05     int aux;
06     aux = x;
07     x = y;
08     y = aux;
09 }

10 main()
11 {
12     int a, b;
13     printf("Escreva um numero para A ----> ");
14     scanf("%d", &a);
15     printf("Escreva outro numero para B -> ");
16     scanf("%d", &b);
17     troca(a, b);
18     printf("Os valores finais para A e B sao %d e %d.", a, b);
19     getch();
20 }
```

# Ponteiros

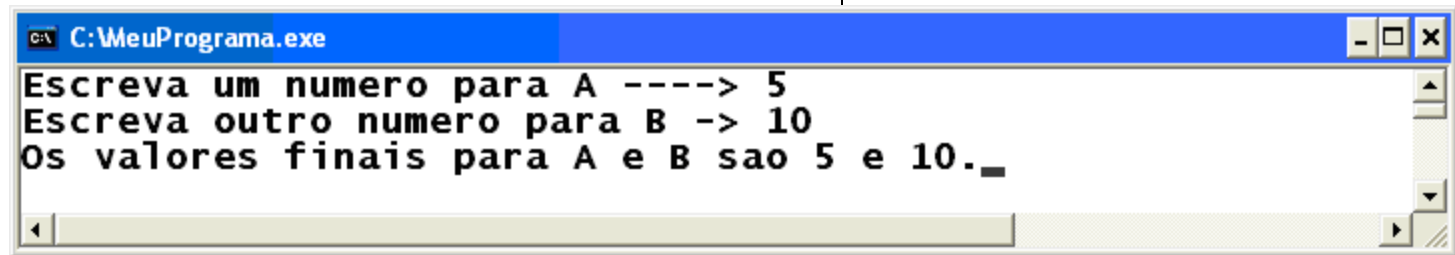
## ■ Passagem de argumentos por valor

```
01 #include <stdio.h>
02 #include <conio.h>

03 void troca(int x, int y)
04 {
05     int aux;
06     aux = x;
07     x = y;
08     y = aux;
09 }

10 main()
11 {
12     int a, b;
13     printf("Escreva um numero para A ----> ");
14     scanf("%d", &a);
15     printf("Escreva outro numero para B -> ");
16     scanf("%d", &b);
17     troca(a, b);
18     printf("Os valores finais para A e B sao %d e %d.", a, b);
19     getch();
20 }
```

*Funciona?*



```
C:\MeuPrograma.exe
Escreva um numero para A ----> 5
Escreva outro numero para B -> 10
Os valores finais para A e B sao 5 e 10._
```

# Ponteiros

## ■ Passagem de argumentos por valor

```
01 #include <stdio.h>
02 #include <conio.h>
```

```
03 void troca(int x, int y)
04 {
05     int aux;
06     aux = x;
07     x = y;
08     y = aux;
09 }
```

```
10 main()
11 {
12     int a, b;
13     printf("Escreva um numero para A ----> ");
14     scanf("%d", &a);
15     printf("Escreva outro numero para B -> ");
16     scanf("%d", &b);
17     troca(a, b);
18     printf("Os valores finais para A e B sao %d e %d.", a, b);
19     getch();
20 }
```

**Ordem de  
execução:**

**1, 2, 10, 11, 12, 13,  
14, 15, 16, 17, 3, 4, 5,  
6, 7, 8, 9, 18, 19, 20**

**Funcionamento  
da memória:**

Identificador	Posição	Valor
a	528	5
b	524	10
x	520	15
y	516	15
aux	512	5

# Ponteiros

## ■ Argumentos valor X argumentos referência

```
01 #include <stdio.h>
02 #include <conio.h>

03 void troca(int x, int y)
04 {
05     int aux;
06     aux = x;
07     x = y;
08     y = aux;
09 }

10 main()
11 {
12     int a, b;
13     printf("Escreva um numero para A -->");
14     scanf("%d", &a);
15     printf("Escreva outro numero para B ->");
16     scanf("%d", &b);
17     troca(a, b);
18     printf("Os valores finais para A e B
           sao %d e %d.", a, b);

19     getch();
20 }
```

```
01 #include <stdio.h>
02 #include <conio.h>

03 void troca(int *x, int *y)
04 {
05     int aux;
06     aux = *x;
07     *x = *y;
08     *y = aux;
09 }

10 main()
11 {
12     int a, b;
13     printf("Escreva um numero para A --->");
14     scanf("%d", &a);
15     printf("Escreva outro numero para B -> ");
16     scanf("%d", &b);
17     troca(&a, &b);
18     printf("Os valores finais para A e B
           sao %d e %d.", a, b);

19     getch();
20 }
```

Os parâmetros *x* e *y* são ponteiros para valores do tipo *int*.

# Ponteiros

## ■ Passagem de argumentos por referência

```
01 #include <stdio.h>
02 #include <conio.h>
```

```
03 void troca(int *x, int *y)
```

```
04 {
```

```
05     int aux;
```

```
06     aux = *x;
```

```
07     *x = *y;
```

```
08     *y = aux;
```

```
09 }
```

```
s
```

```
10 main()
```

```
11 {
```

```
12     int a, b;
```

```
13     printf("Escreva um numero para A ---> ");
```

```
14     scanf("%d", &a);
```

```
15     printf("Escreva outro numero para B -> ");
```

```
16     scanf("%d", &b);
```

```
17     troca(&a, &b);
```

```
18     printf("Os valores finais para A e B sao  
           %d e %d.", a, b);
```

```
19     getch();
```

```
20 }
```

**Ordem de  
execução:**

1, 2, 10, 11, 12, 13,  
14, 15, 16, 17, 3, 4, 5,  
6, 7, 8, 9, 18, 19, 20

**Funcionamento  
da memória:**

Identificador	Posição	Valor
a	528	15
b	524	15
x	520	528
y	516	524
aux	512	5

# Ponteiros

## ■ Passagem de argumentos por referência

*Chegou o momento de explicar porque precisamos usar o operador ‘&’ ao usar o comando “scanf”!!!*

*“scanf” precisa MODIFICAR o valor de um identificador, logo este identificador deve ser passado por referência!!!*

```
scanf("%d", &numero);
```

---

# Ponteiros

- Exercícios do capítulo 3

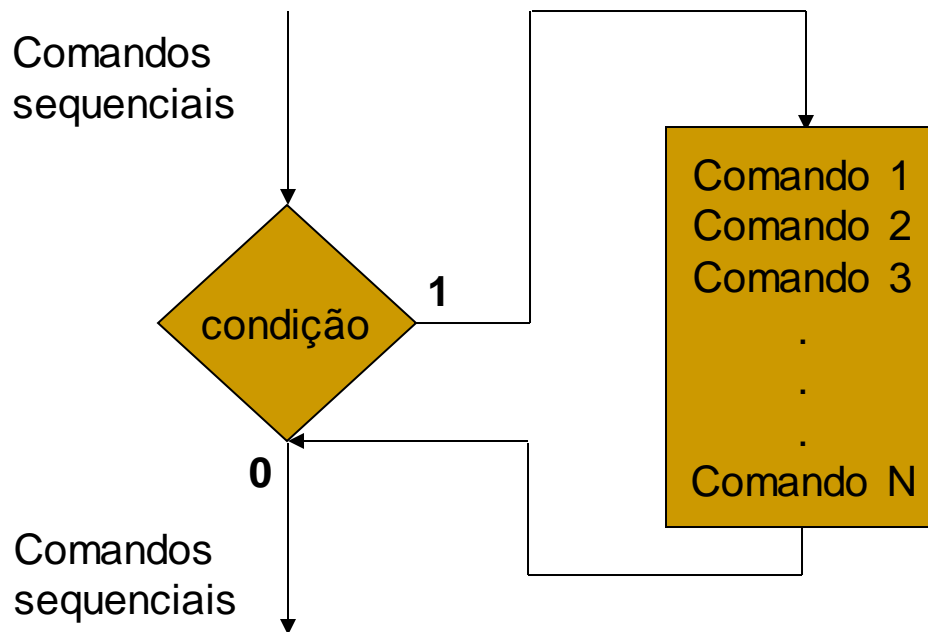




# Estruturas de seleção

## ■ Comando *if*

- Fluxograma / sintaxe
- **A condição deve retornar um valor BOOLEANO**



//comandos sequenciais

```
if (condição)
{
    //comando 1
    //comando 2
    //comando 3
    //...
    //comando N
}
```

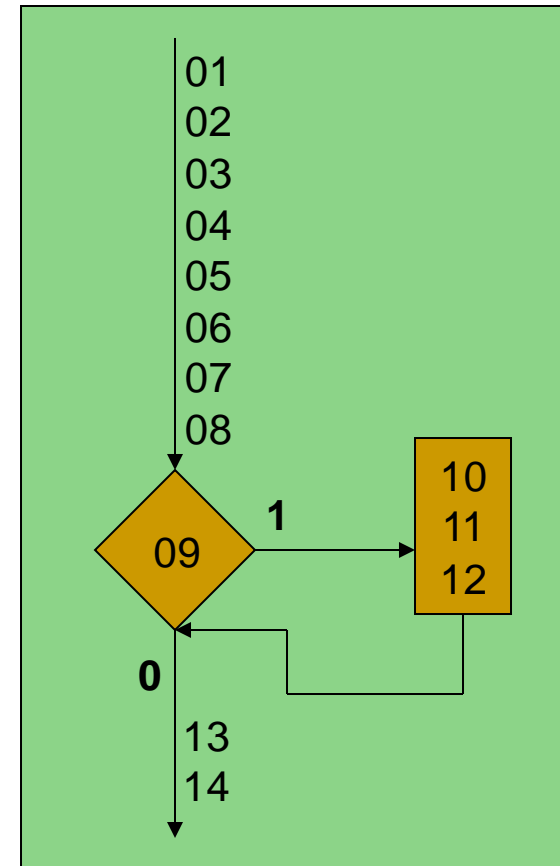
//comandos sequenciais

# Estruturas de seleção

## ■ Comando *if*

### □ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int N;
06     printf("Programa que le um inteiro
           e o exhibe se for positivo\n\n");
07     printf("Escreva um numero inteiro: ");
08     scanf("%d", &N);
09     if (N > 0)
10     {
11         printf("%d", N);
12     }
13     getch();
14 }
```

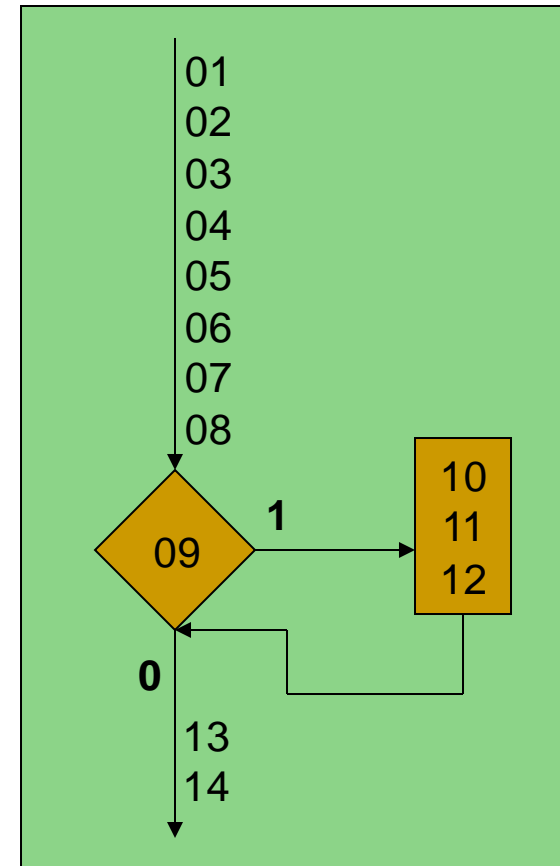


# Estruturas de seleção

## ■ Comando *if*

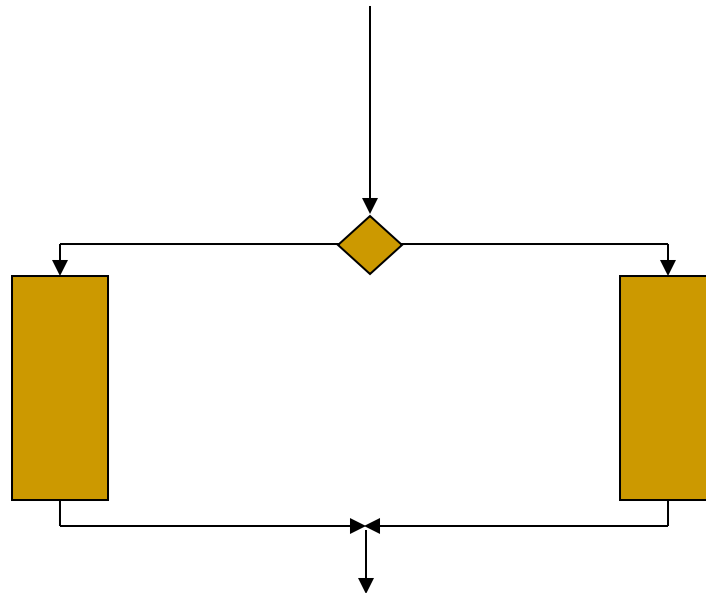
### □ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int N;
06     printf("Programa que le um inteiro
           e o exibe se for positivo\n\n");
07     printf("Escreva um numero inteiro: ");
08     scanf("%d", &N);
09     if (N > 0)
10     {
11         printf("%d", N);
12     }
13     getch();
14 }
```



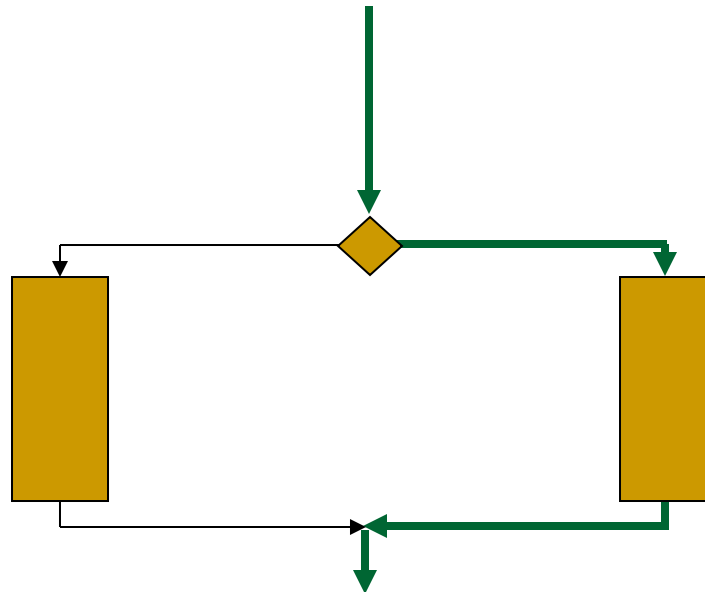
# Estruturas de seleção

- Comando *if* com cláusula *else*
  - Sempre desviamos o processamento
  - Resta saber por qual caminho



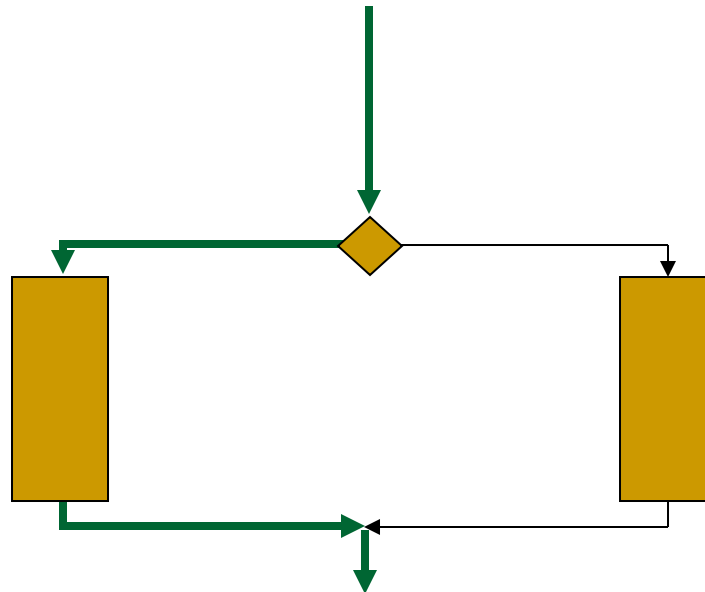
# Estruturas de seleção

- Comando *if* com cláusula *else*
  - Sempre desviamos o processamento
  - Resta saber por qual caminho



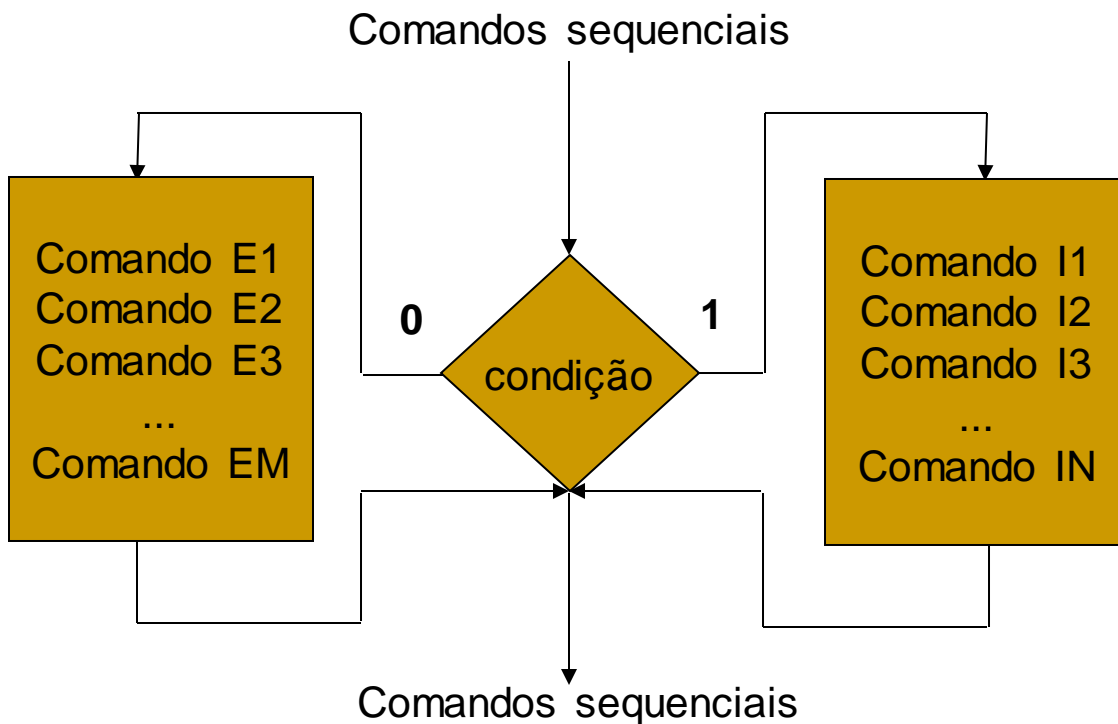
# Estruturas de seleção

- Comando *if* com cláusula *else*
  - Sempre desviamos o processamento
  - Resta saber por qual caminho



# Estruturas de seleção

- Comando *if* com cláusula *else*
  - Fluxograma / sintaxe
  - **A condição deve retornar um valor BOOLEANO**

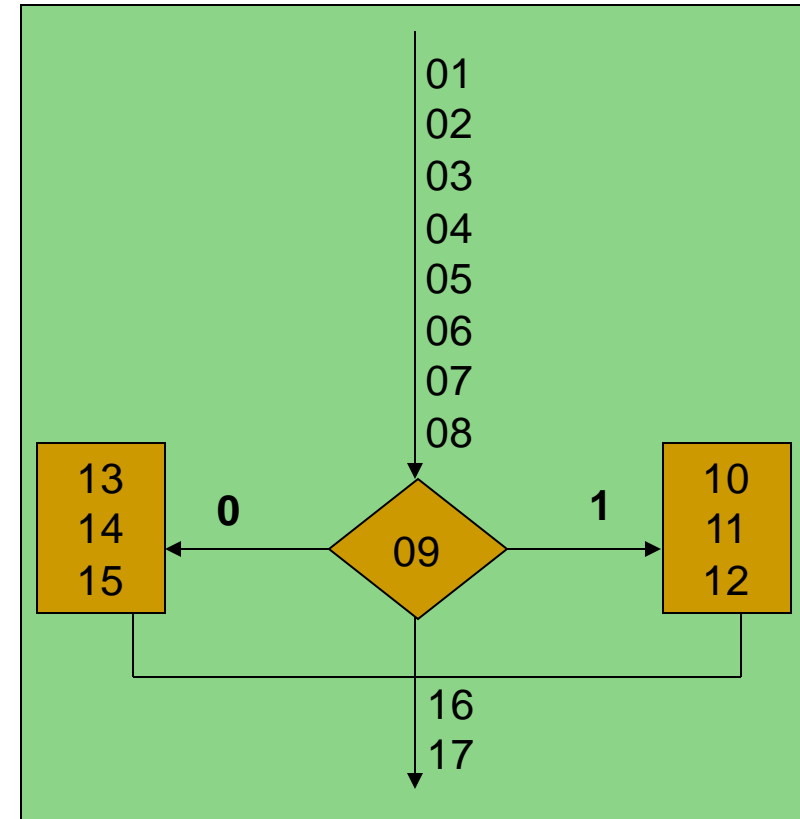


```
//comandos sequenciais
if (condição)
{
    //comando I1
    //comando I2
    //comando I3
    //...
    //comando IN
}
else {
    //comando E1
    //comando E2
    //comando E3
    //...
    //comando EM
}
//comandos sequenciais
```

# Estruturas de seleção

- Comando *if* com cláusula *else*
  - Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int N;
06     printf("Programa que le um inteiro e diz se
           e ou nao maior que zero\n\n");
07     printf("Escreva um numero inteiro: ");
08     scanf("%d", &N);
09     if (N > 0)
10     {
11         printf("%d e maior que zero", N);
12     }
13     else {
14         printf("%d nao e maior que zero", N);
15     }
16     getch();
17 }
```

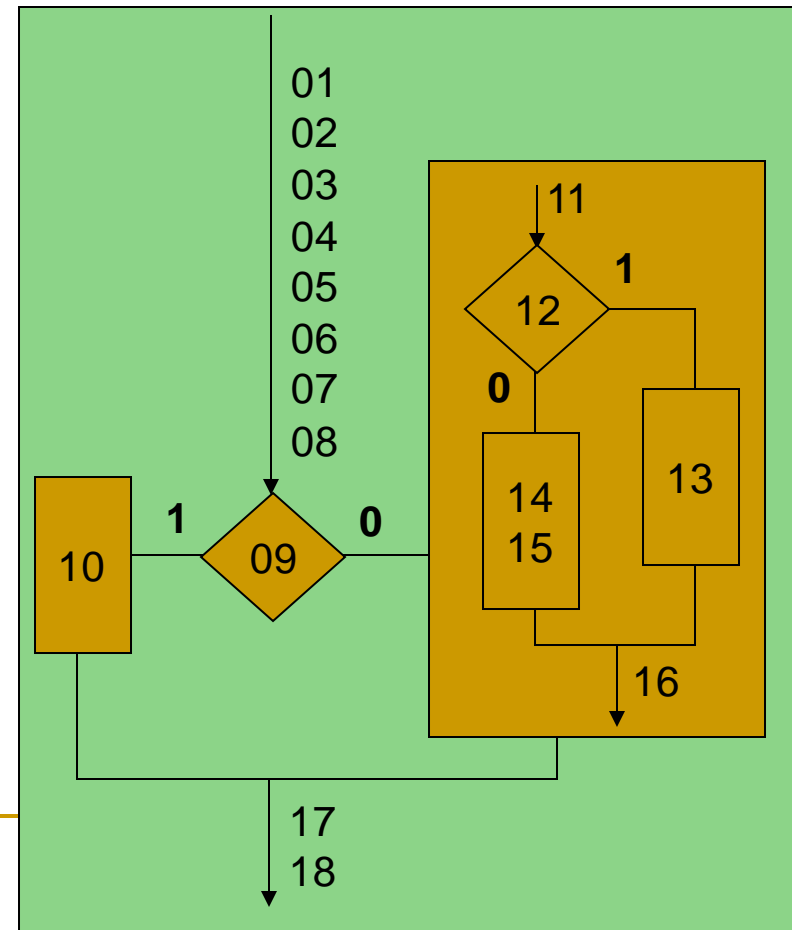




# Estruturas de seleção

- Aninhamento de comandos *if*
  - *if* dentro de um *if* ou de um *else*

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int N;
06     printf("Programa que ve se um inteiro e maior,
           menor ou igual a zero\n\n");
07     printf("Escreva um numero inteiro: ");
08     scanf("%d", &N);
09     if (N > 0)
10         printf("%d e maior que zero", N);
11     else {
12         if (N < 0)
13             printf("%d e menor que zero", N);
14         else
15             printf("%d e igual a zero", N);
16     }
17     getch();
18 }
```



# Estruturas de seleção

## ■ O comando *switch*

### □ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     float X, Y;
06     char OP;
07     const char * cabecalho = "Simulador de uma calculadora basica
                                de numeros reais\n\n";
08     printf ("%s" , cabecalho);
09     printf("Digite o primeiro operando: ");
10     scanf("%f", &X);
11     printf("Digite o segundo operando: ");
12     scanf("%f", &Y);
13     printf("Digite o operador: ");
14     scanf("\n%c", &OP);
15     //O '\n' serve para consumir o <ENTER> digitado na entrada da
                                segunda variável float
```

# Estruturas de seleção

## ■ O comando *switch*

### □ Exemplo

```
15  switch (OP)
16  {
17      case '+' : printf("Resultado: %f", X+Y);
18              break;
19      case '-' : printf("Resultado: %f", X-Y);
20              break;
21      case '*' :
22      case 'x' :
23      case 'X' : printf("Resultado: %f", X*Y);
24              break;
25      case '/' : printf("Resultado: %f", X/Y);
26              break;
27      default : printf("\nOperador invalido (%c).", OP);
28  }
29  getch();
30 }
```

# Estruturas de seleção

## ■ O comando *switch*

### □ Sintaxe

- A expressão é um escalar (*int* ou *char*)
- O *break* é opcional faz o processamento seguir para fora do *switch*
- O *default* é opcional

```
switch (expressão)
{
    case constante1 : comando1DaConstante1;
                    comando2DaConstante1;
                    ...
                    comandoWDaConstante1;
                    break;

    case constante2 : comando1DaConstante2;
                    comando2DaConstante2;
                    ...
                    comandoXDaConstante2;
                    break;

    ...

    case constanteN : comando1DaConstanteN;
                    comando2DaConstanteN;
                    ...
                    comandoYDaConstanteN;
                    break;

    default          : comando1DoDefault;
                    comando2DoDefault;
                    ...
                    comandoZDoDefault;
}
```

# Estruturas de seleção

## ■ O comando *switch*

- switch X aninhamento de comandos *if*

```
switch (OP)
{
    case '+' : printf("Resultado: %f", X+Y);
               break;
    case '-' : printf("Resultado: %f", X-Y);
               break;
    case '*' :
    case 'x' :
    case 'X' : printf("Resultado: %f", X*Y);
               break;
    case '/' : printf("Resultado: %f", X/Y);
               break;
    default  : printf("\nOperador invalido (%c).", OP);
}
}
```

# Estruturas de seleção

## ■ O comando *switch*

### □ switch X aninhamento de comandos *if*

```
if (OP == '+')
    printf("Resultado: %f", X+Y);
else if (OP == '-')
    printf("Resultado: %f", X-Y);
else if ((OP == '*') || (OP == 'x') || (OP == 'X'))
    printf("Resultado: %f", X*Y);
else if (OP == '/')
    printf("Resultado: %f", X/Y);
else printf("\nOperador invalido (%c).", OP);
```

---

# Estruturas de seleção

- O operador ternário

condição ? expressão1 : expressão2

# Estruturas de seleção

## ■ O operador ternário

condição ? expressão1 : expressão2

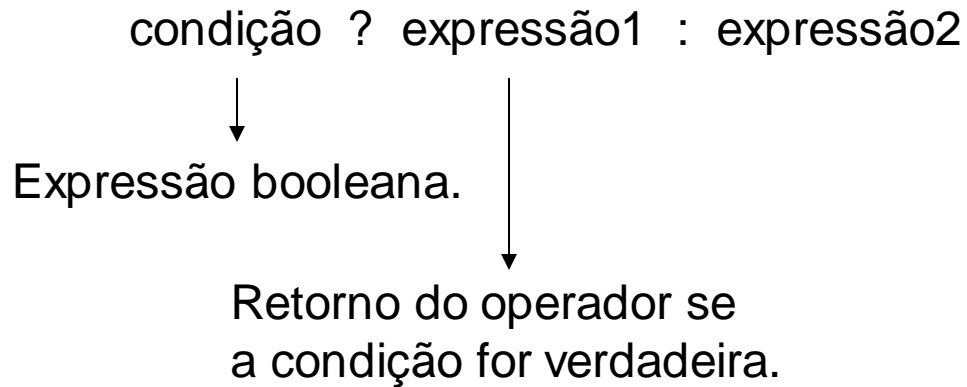


Expressão booleana.



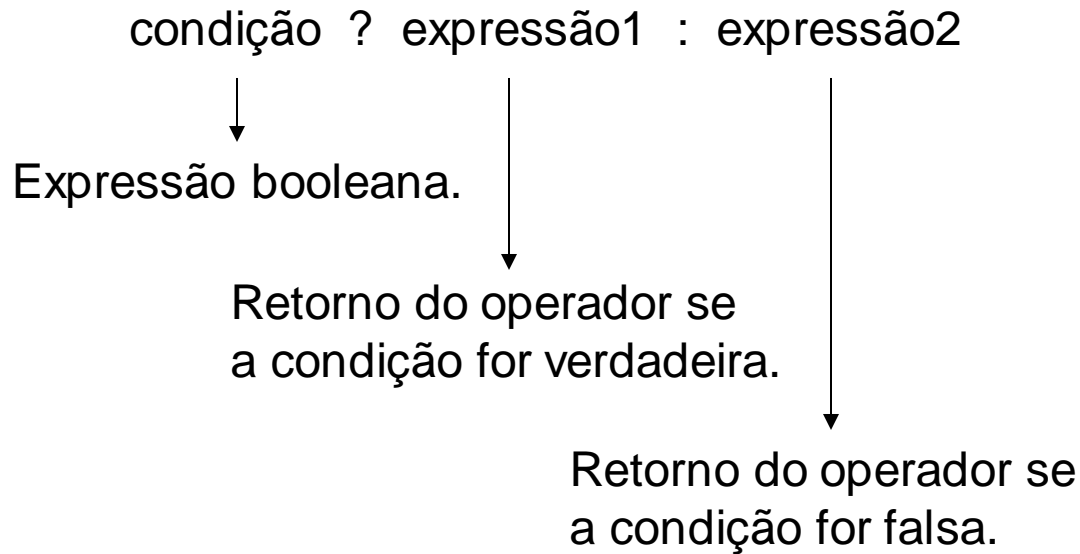
# Estruturas de seleção

## ■ O operador ternário



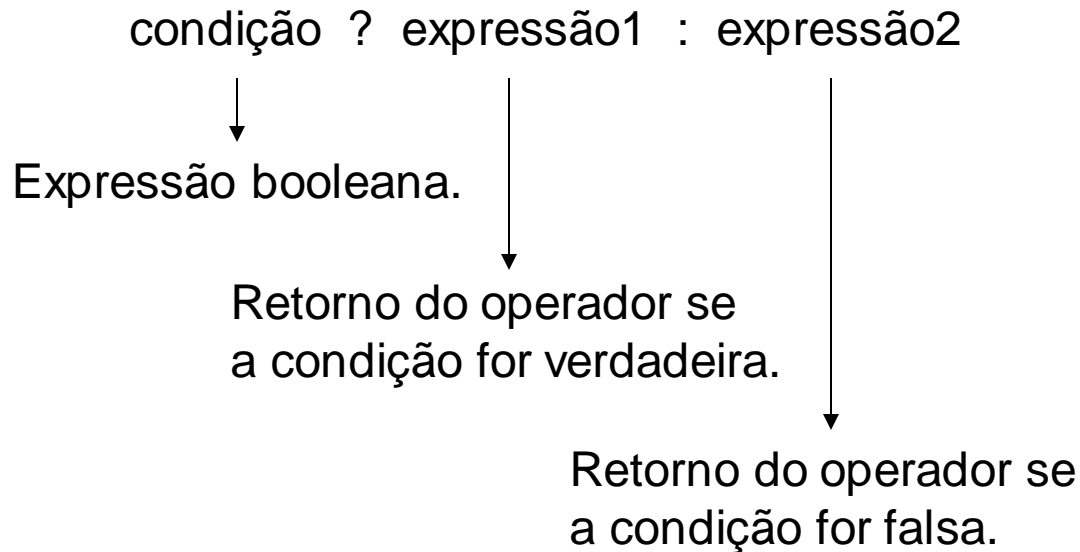
# Estruturas de seleção

## ■ O operador ternário



# Estruturas de seleção

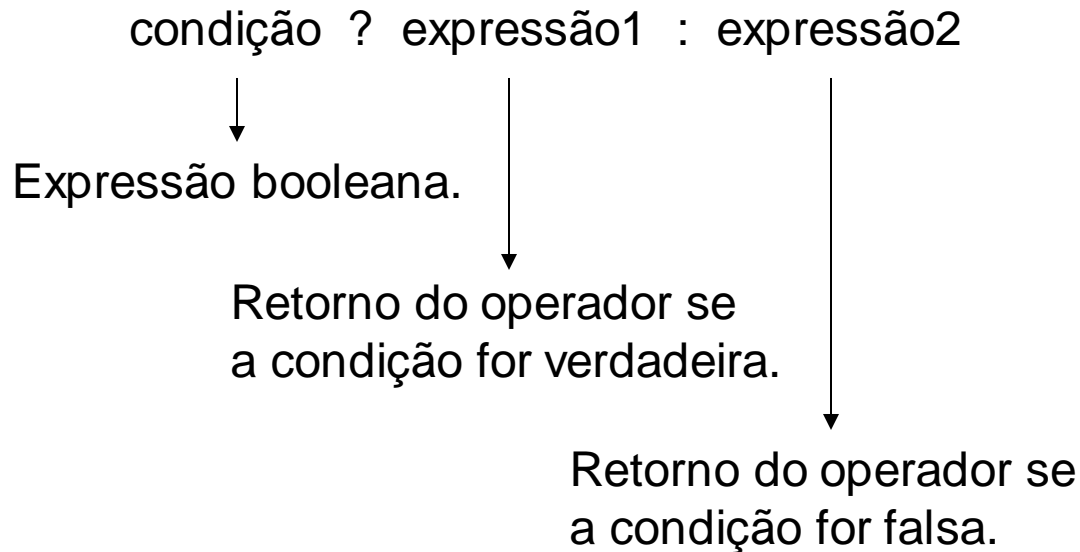
## ■ O operador ternário



Substitui o comando *if* quando os blocos do *if* e também do *else* consistem de uma única expressão.

# Estruturas de seleção

## ■ O operador ternário



Substitui o comando *if* quando os blocos do *if* e também do *else* consistem de uma única expressão.

```
printf("%s", x==y ? "iguais" : "diferentes");
```

---

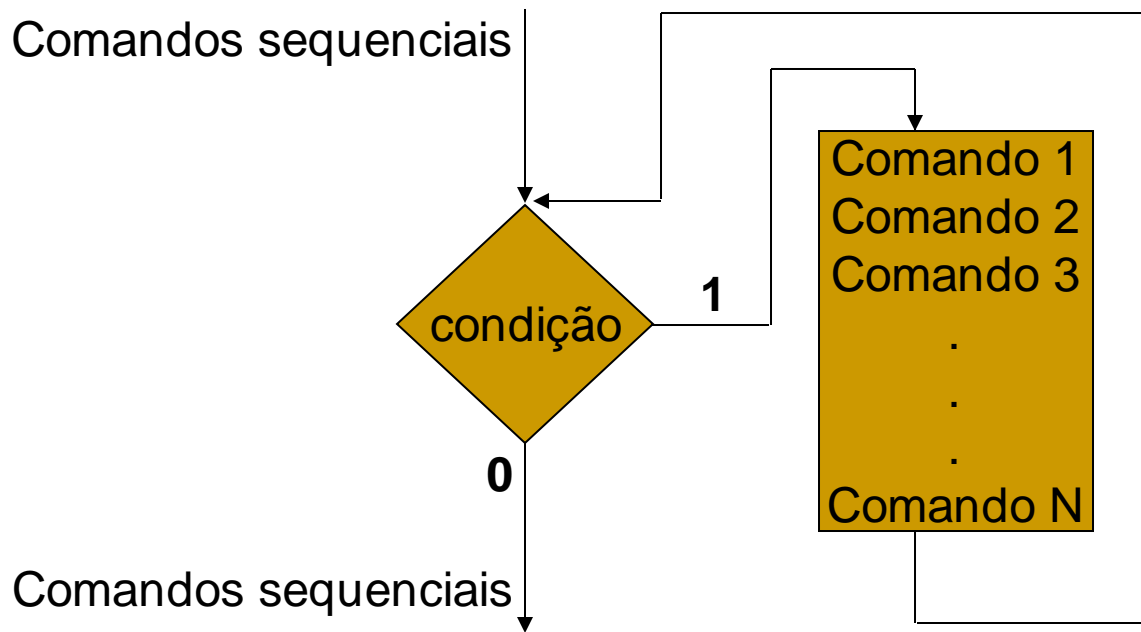
# Estruturas de seleção

- Exercícios do capítulo 4



# Estruturas de repetição

- O comando *while*
  - Fluxograma / sintaxe



//comandos sequenciais

**while** (condição)

{

  //comando 1

  //comando 2

  //comando 3

  //...

  //comando N

}

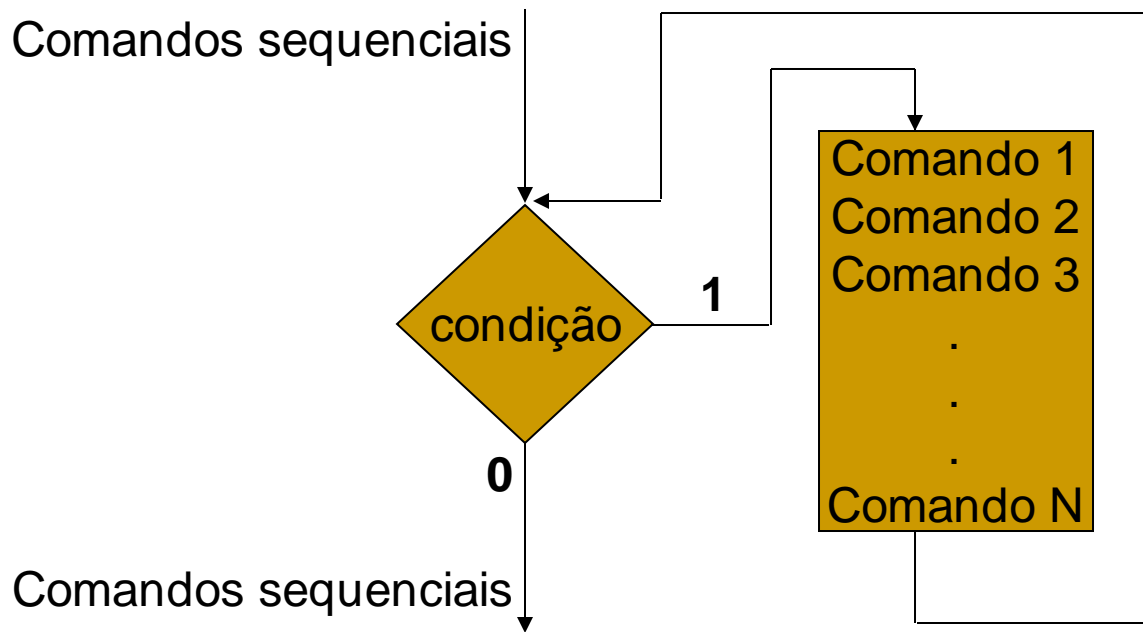
//comandos sequenciais

# Estruturas de repetição

## ■ O comando *while*

### □ Fluxograma / sintaxe

- A condição deve envolver **variável(is) de controle** e retornar um valor *booleano*



//comandos sequenciais

**while** (condição)

{

  //comando 1

  //comando 2

  //comando 3

  //...

  //comando N

}

//comandos sequenciais

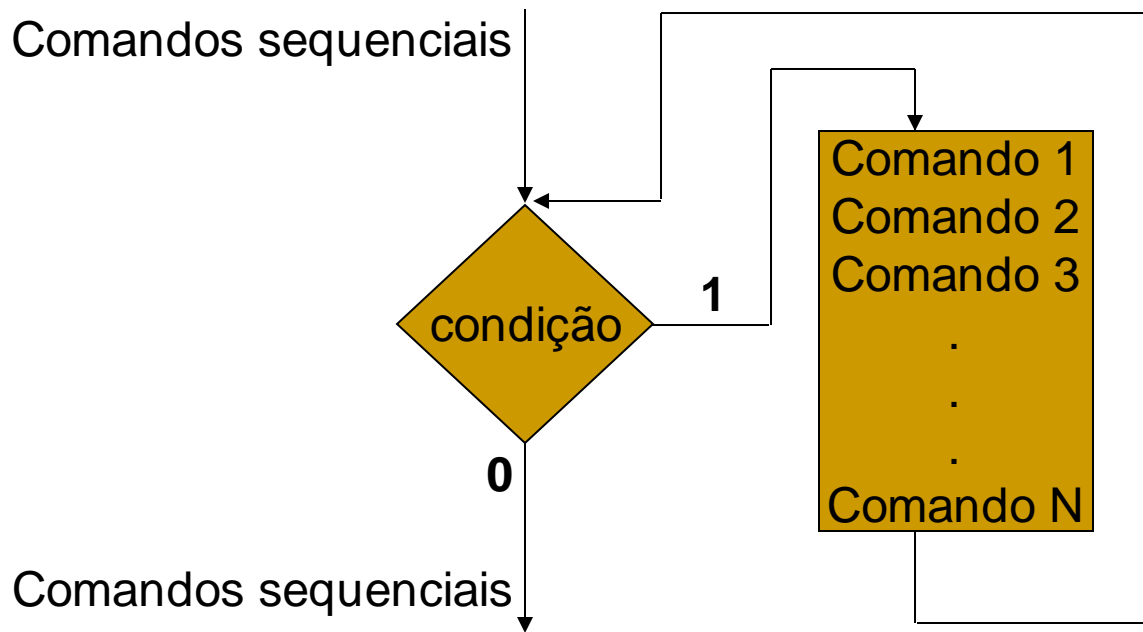


# Estruturas de repetição

## ■ O comando *while*

### □ Fluxograma / sintaxe

- A(s) **variável(is) de controle** da condição deve(m) ser iniciada(s) antes do laço



//comandos sequenciais

**while** (condição)

{

  //comando 1

  //comando 2

  //comando 3

  //...

  //comando N

}

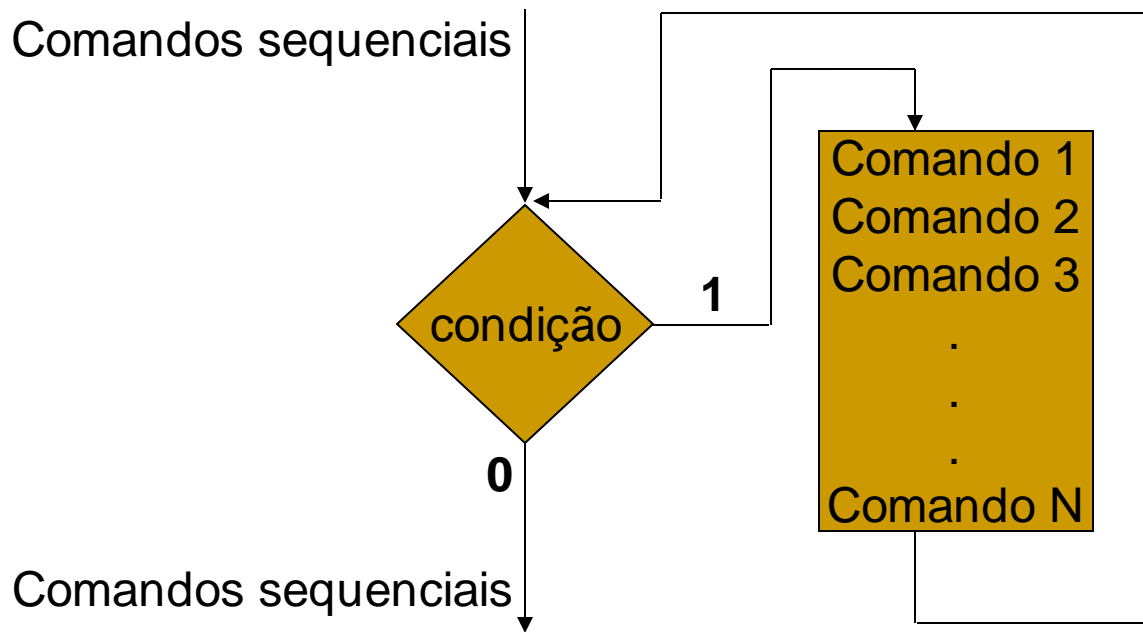
//comandos sequenciais

# Estruturas de repetição

## ■ O comando *while*

### □ Fluxograma / sintaxe

- No corpo do laço, a(s) **variável(is) de controle** da condição deve(m) ser atualizada(s)



//comandos sequenciais

**while** (condição)

{

  //comando 1

  //comando 2

  //comando 3

  //...

  //comando N

}

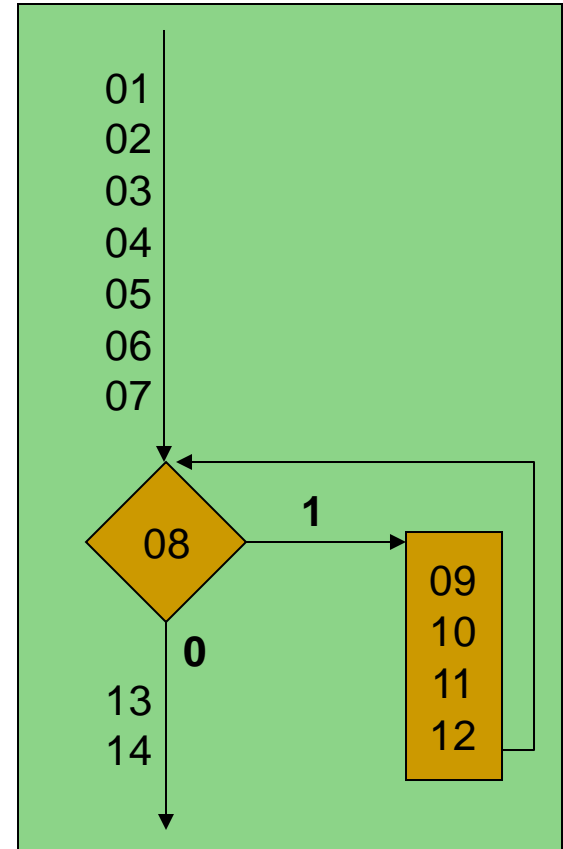
//comandos sequenciais

# Estruturas de repetição

## ■ O comando *while*

### □ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int N;
06     printf("Impressão de CEFET 10 vezes:\n\n");
07     N = 1;
08     while (N <= 10)
09     {
10         printf("CEFET\n");
11         N = N + 1; //++N;
12     }
13     getch();
14 }
```

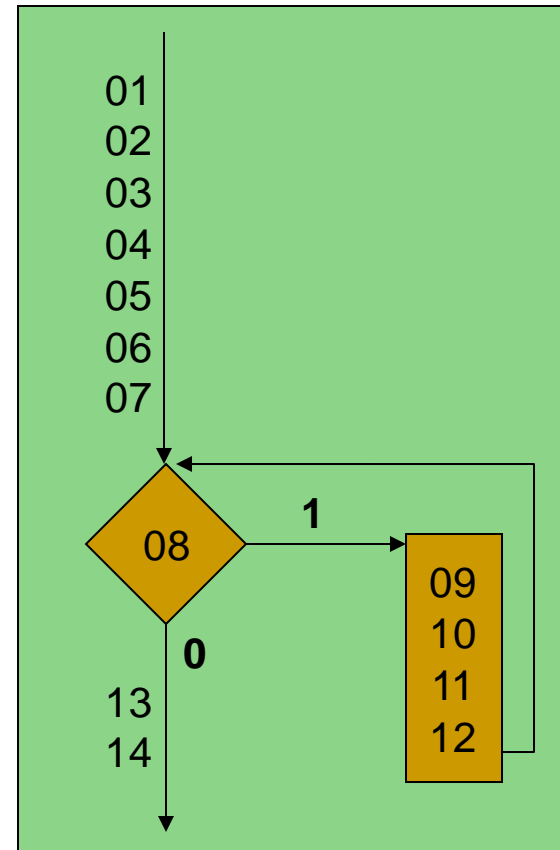


# Estruturas de repetição

## ■ O comando *while*

### □ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int N;
06     printf("Lista dos numeros entre 1 e 100:\n\n");
07     N = 1;
08     while (N <= 100)
09     {
10         printf("%8d", N);
11         N = N + 1; //++N;
12     }
13     getch();
14 }
```

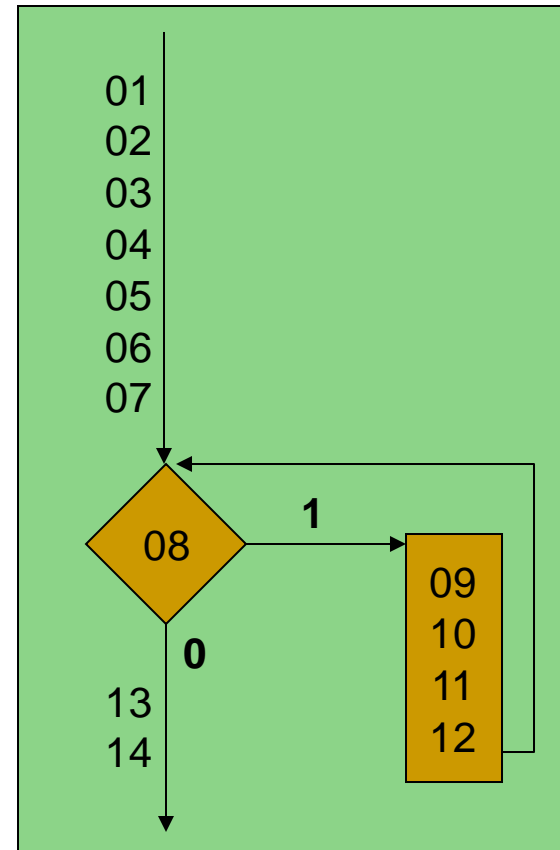


# Estruturas de repetição

## ■ O comando *while*

### □ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     char C;
06     printf("Impressao do alfabeto:\n\n");
07     C = 'A';
08     while (C <= 'Z')
09     {
10         printf("%c\n", C);
11         C = C + 1; //++C;
12     }
13     getch();
14 }
```



# Estruturas de repetição

- O comando *for*
  - ❑ Substitui o comando *while*
  - ❑ Sintaxe menor

```
variávelDeControle = vi;
```

```
while (condição)
```

```
{
```

```
    // comandos
```

```
    variávelDeControle = novoValor;
```

```
}
```

```
for (variávelDeControle = vi; condição; variávelDeControle = novoValor)
```

```
{
```

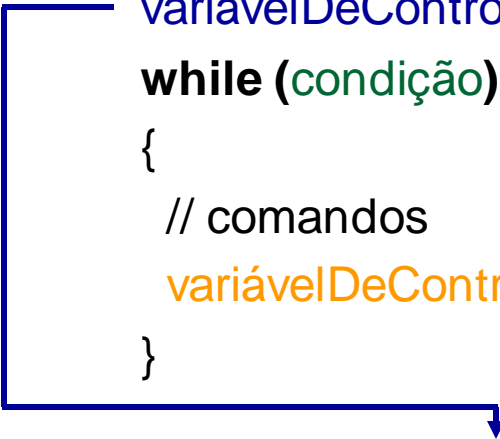
```
    // comandos
```

```
}
```

# Estruturas de repetição

- O comando *for*
  - ❑ Substitui o comando *while*
  - ❑ Sintaxe menor

```
variávelDeControle = vi;  
while (condição)  
{  
    // comandos  
    variávelDeControle = novoValor;  
}
```



```
for (variávelDeControle = vi; condição; variávelDeControle = novoValor)  
{  
    // comandos  
}
```

# Estruturas de repetição

- O comando *for*
  - ❑ Substitui o comando *while*
  - ❑ Sintaxe menor

```
variávelDeControle = vi;
```

```
while (condição)
```

```
{
```

```
    // comandos
```

```
    variávelDeControle = novoValor;
```

```
}
```

```
for (variávelDeControle = vi; condição; variávelDeControle = novoValor)
```

```
{
```

```
    // comandos
```

```
}
```



# Estruturas de repetição

- O comando *for*
  - ❑ Substitui o comando *while*
  - ❑ Sintaxe menor

```
variávelDeControle = vi;
```

```
while (condição)
```

```
{
```

```
    // comandos
```

```
    variávelDeControle = novoValor;
```

```
}
```

```
for (variávelDeControle = vi; condição; variávelDeControle = novoValor)
```

```
{
```

```
    // comandos
```

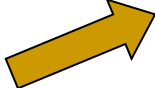
```
}
```

# Estruturas de repetição

## ■ O comando *for*


### □ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int N;
06     printf("Lista dos numeros entre 1 e 100\n\n");
07     N = 1;
08     while (N <= 100)
09     {
10         printf("%8d", N);
11         N = N + 1;
12     }
13     getch();
14 }
```



com "while"

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int N;
06     printf("Lista dos números entre 1 e 100\n\n");
07     for (N = 1; N <= 100; N = N + 1)
08     {
09         printf("%8d", N);
10     }
11     getch();
12 }
```



com "for"

---

# Estruturas de repetição

- Saída forçada (*break*)
  - Recomendado apenas dentro de loops infinitos

# Estruturas de repetição

- Saída forçada (*break*)
  - Recomendado apenas dentro de loops infinitos
  - Exemplo

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
main()
{
    char opcao;
    while (1) //loop infinito
    {
        system("cls");
        //comandos do programa
        printf("Deseja repetir o programa: [S|N]: ");
        scanf("\n%c", &opcao);
        if (opcao == 'n' || opcao == 'N') break;
    }
    system("pause");
}
```

# Estruturas de repetição

- Saída forçada (*break*)
  - Recomendado apenas dentro de loops infinitos
  - Exemplo

O comando *system*, do arquivo de cabeçalho *stdlib.h* recebe um texto como parâmetro, o qual é entendido como um comando do sistema operacional no qual o programa será executado.

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
main()
{
    char opcao;
    while (1) //loop infinito
    {
        system("cls");
        //comandos do programa
        printf("Deseja repetir o programa: [S|N]: ");
        scanf("\n%c", &opcao);
        if (opcao == 'n' || opcao == 'N') break;
    }
    system("pause");
}
```

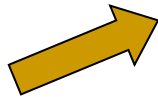
# Estruturas de repetição

## ■ Saída forçada (*break*)

- Recomendado apenas dentro de *loops* infinitos
- Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int N;
06     N = 1;
07     while (N <= 100)
08     {
09         printf("%8d", N);
10         N = N + 1;
11     }
12     getch();
13 }
```

com "while" comum



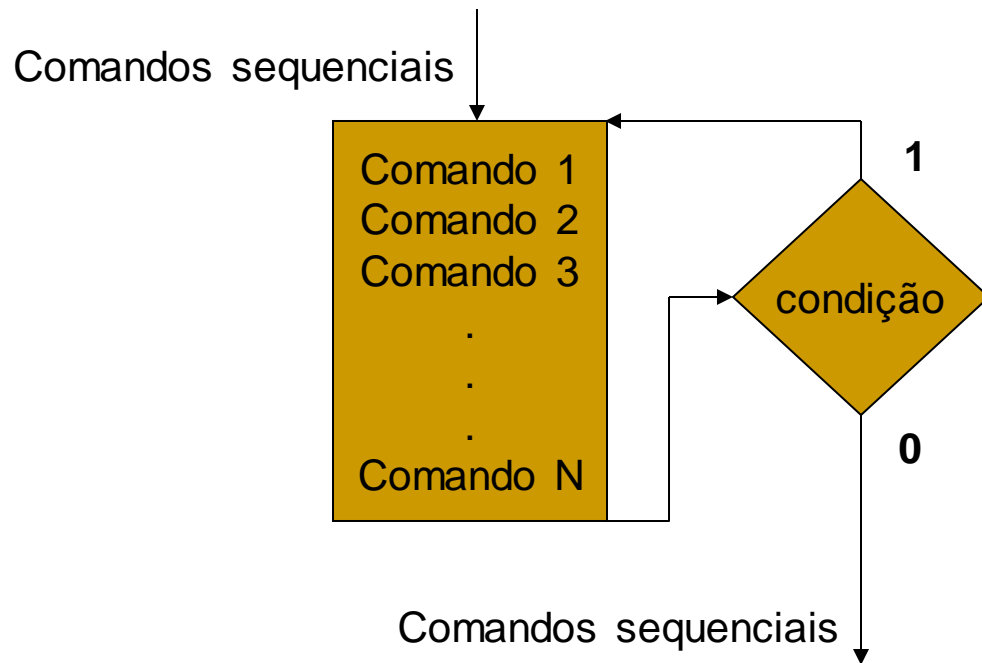
```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int N;
06     N = 1;
07     while (1)
08     {
09         printf("%8d", N);
10         if (N == 100) break;
11         N = N + 1;
12     }
13     getch();
14 }
```

com saída forçada



# Estruturas de repetição

- O comando *do while*
  - Fluxograma / sintaxe
    - A condição deve retornar um valor *booleano*



//comandos sequenciais

```
do  
{  
  //comando 1  
  //comando 2  
  //comando 3  
  //...  
  //comando N  
}
```

```
while (condição);
```

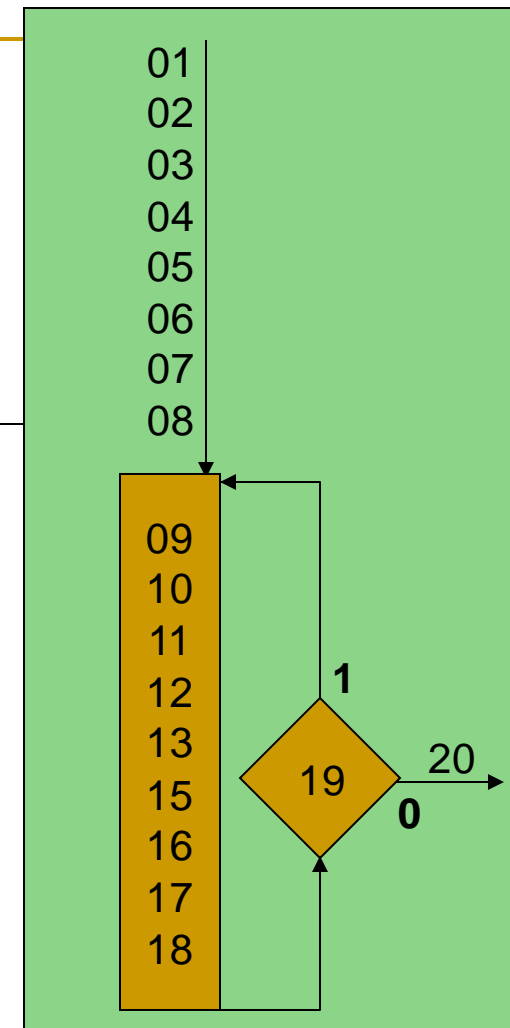
//comandos sequenciais

# Estruturas de repetição

## ■ O comando *do while*

### □ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 #include <stdlib.h>
04 main()
05 {
06     float N1, N2, MEDIA;
07     char RESP;
08     do
09     {
10         system("cls");
11         printf("Programa para calcular a media entre dois numeros:\n\n");
12         printf("Digite os dois numeros separados por espaco: ");
13         scanf("%f %f", &N1, &N2);
14         MEDIA = (N1+N2)/2;
15         printf("A media entre %f e %f e %f.\n\n", N1, N2, MEDIA);
16         printf("Deseja repetir o programa? Entre [S] para sim ou outra tecla para nao: ");
17         scanf("\n%c", &RESP);
18     }
19     while ((RESP == 'S') || (RESP == 's'));
20 }
```





# Estruturas de repetição

## ■ Recursividade

- Uma função é definida em termos de si mesma
- Estabelece uma relação de recorrência que substitui um laço de repetição
- Deve apresentar um critério de parada para que não resulte em um *loop* infinito.

# Estruturas de repetição

## ■ Recursividade

### □ Exemplo

```
int fatorial ( int n )  
{  
    if (n == 0) return 1;  
    else return (n * fatorial(n-1));  
}
```

# Estruturas de repetição

## ■ Recursividade

### □ Exemplo

```
int fatorial ( int n )  
{  
    if (n == 0) return 1;  
    else return (n * fatorial(n-1));  
}
```

fatorial(3) = 3 x fatorial(2)

fatorial(2) = 2 x fatorial(1)

fatorial(1) = 1 x fatorial(0)

fatorial(0) = 1

fatorial(3) = 3 x 2 x 1 x 1 = 6

# Estruturas de repetição

- Recursividade
  - Outro exemplo

```
int fibo(int n)
{
    if (n == 1) return 0;
    else if (n == 2) return 1;
    else return fibo(n-1) + fibo(n-2);
}
```

---

# Estruturas de repetição

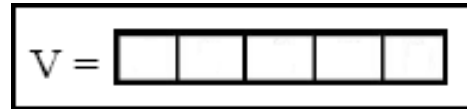
- Exercícios do capítulo 5



# Arranjos

- Vetor

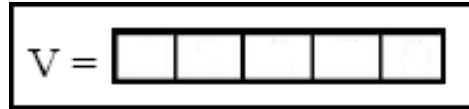
- Estrutura



# Arranjos

## ■ Vetor

□ Estrutura



□ Declaração de variável vetor

tipo nomeDaVariável[númeroDeColunas];

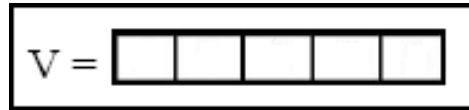
int V[5];



# Arranjos

## ■ Vetor

□ Estrutura



□ Declaração de variável vetor

tipo nomeDaVariável[númeroDeColunas];

int V[5];

Declaração de vários vetores de um mesmo tipo:

tipo nome1[tam1], ..., nomeN[tamN];

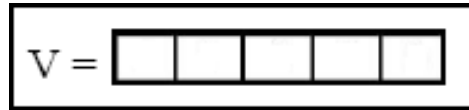
Exemplo:

int X[5], Y[10], Z[15];

# Arranjos

## ■ Vetor

□ Estrutura



□ Declaração de variável vetor

tipo nomeDaVariável[númeroDeColunas];

int V[5];

Declaração de vários vetores de um mesmo tipo:

tipo nome1[tam1], ..., nomeN[tamN];

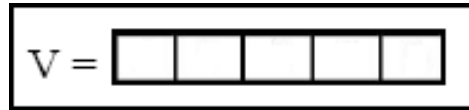
Exemplo:

int X[5], Y[10], Z[15];

# Arranjos

## ■ Vetor

□ Estrutura



□ Declaração de variável vetor

tipo nomeDaVariável[númeroDeColunas];

int V[5];

Declaração de vários vetores de um mesmo tipo:

tipo nome1[tam1], ..., nomeN[tamN];

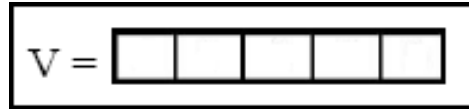
Exemplo:

int X[5], Y[10], Z[15];

# Arranjos

## ■ Vetor

### □ Estrutura



### □ Declaração de variável vetor

tipo nomeDaVariável[númeroDeColunas];

int V[5];

### □ Acesso

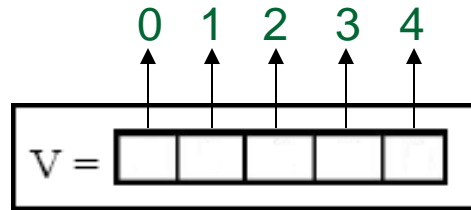
nomeDaVariável[coluna];

V[0] = 4;   V[1] = 7;   V[2] = 2;   V[3] = 5;   V[4] = 3;

# Arranjos

## ■ Vetor

### □ Estrutura



### □ Declaração de variável vetor

tipo nomeDaVariável[númeroDeColunas];

int V[5];

### □ Acesso

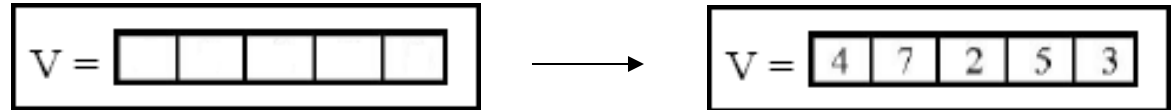
nomeDaVariável[coluna];

V[0] = 4;   V[1] = 7;   V[2] = 2;   V[3] = 5;   V[4] = 3;

# Arranjos

## ■ Vetor

### □ Estrutura



### □ Declaração de variável vetor

tipo nomeDaVariável[númeroDeColunas];

int V[5];

### □ Acesso

nomeDaVariável[coluna];

V[0] = 4;   V[1] = 7;   V[2] = 2;   V[3] = 5;   V[4] = 3;

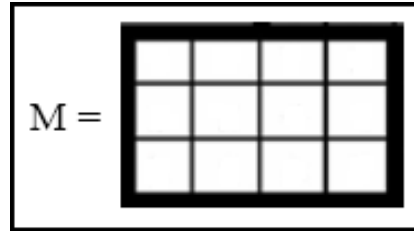
# Arranjos

## ■ Vetor (Exemplo)

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     const int N = 30;
06     int A[N];
07     int I, POS, MAIOR;
08     for (I = 0; I < N; I = I + 1)
09     {
10         printf("Forneca o elemento da posicao %2d do vetor: ", I);
11         scanf("%d", &A[I]);
12     }
13     MAIOR = A[0];
14     POS = 0;
15     for (I = 1; I < N; I = I + 1)
16     {
17         if (A[I] > MAIOR)
18         {
19             MAIOR = A[I];
20             POS = I;
21         }
22     }
23     printf("O maior elemento do vetor e %d e ele esta na posicao %d.", MAIOR, POS);
24     getch();
25 }
```

# Arranjos

- Matriz
- Estrutura



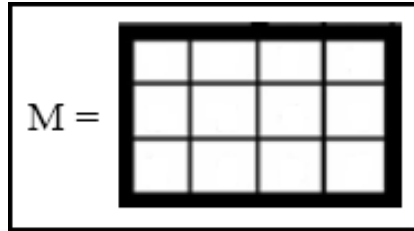


# Arranjos

- Matriz

- Estrutura

- Declaração de variável matriz

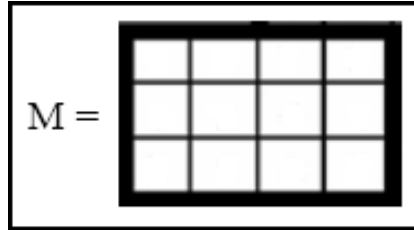


tipo nomeDaVariável[númeroDeLinhas] [númeroDeColunas];  
int M[3][4];

# Arranjos

## ■ Matriz

- Estrutura



- Declaração de variável matriz

tipo nomeDaVariável[númeroDeLinhas] [númeroDeColunas];  
int M[3][4];

Declaração de vários vetores de um mesmo tipo:

tipo nome1[linhas1][colunas1], ..., nomeN[linhasN][colunasN];

Exemplo:

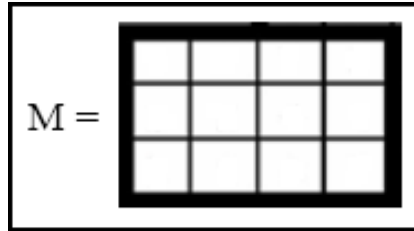
```
int X[2][5], Y[2][3], Z[6][3];
```

# Arranjos

## ■ Matriz

□ Estrutura

□ Declaração de variável matriz



tipo nomeDaVariável[númeroDeLinhas] [númeroDeColunas];  
int M[3][4];

Declaração de vários vetores de um mesmo tipo:

tipo nome1[linhas1][colunas1], ..., nomeN[linhasN][colunasN];

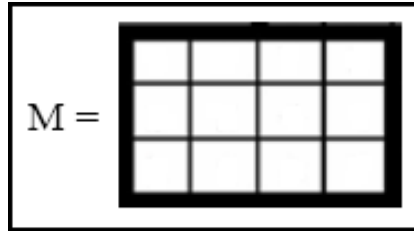
Exemplo:

int X[2][5], Y[2][3], Z[6][3];

# Arranjos

## ■ Matriz

- Estrutura



- Declaração de variável matriz

tipo nomeDaVariável[númeroDeLinhas] [númeroDeColunas];  
int M[3][4];

Declaração de vários vetores de um mesmo tipo:

tipo nome1[linhas1][colunas1], ..., nomeN[linhasN][colunasN];

Exemplo:

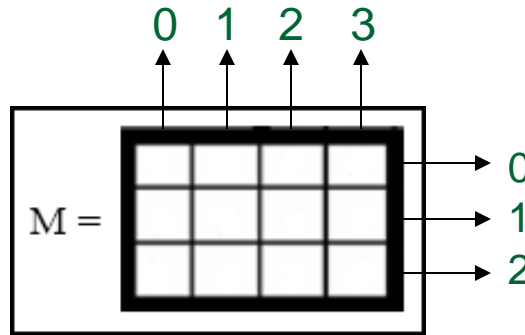
```
int X[2][5], Y[2][3], Z[6][3];
```

# Arranjos

## ■ Matriz

### □ Estrutura

### □ Declaração de variável matriz



tipo nomeDaVariável[númeroDeLinhas] [númeroDeColunas];

int M[3][4];

### □ Acesso

nomeDaVariável[Linha][Coluna]

M[0][0] = 3;

M[0][1] = 8;

M[0][2] = 1;

M[0][3] = 5;

M[1][0] = 0;

M[1][1] = 2;

M[1][2] = 4;

M[1][3] = 7;

M[2][0] = 2;

M[2][1] = 5;

M[2][2] = 9;

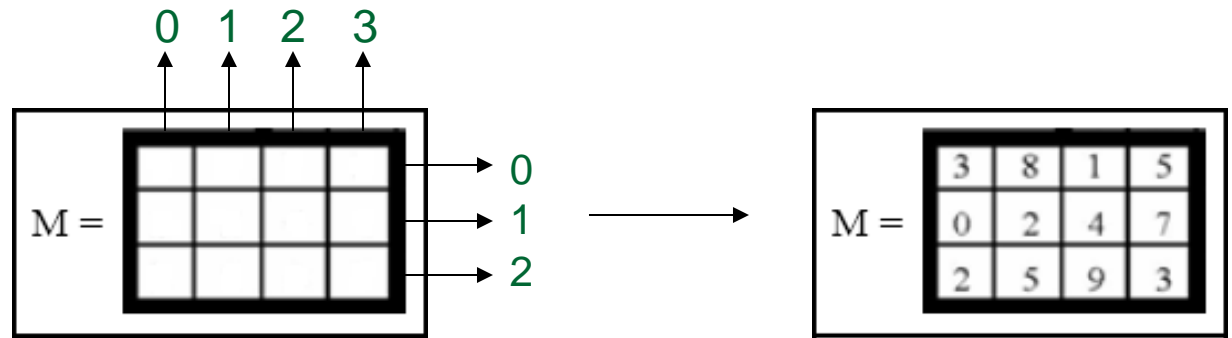
M[2][3] = 3;

# Arranjos

## ■ Matriz

### □ Estrutura

### □ Declaração de variável matriz



tipo nomeDaVariável[númeroDeLinhas] [númeroDeColunas];  
int M[3][4];

### □ Acesso

nomeDaVariável[Linha][Coluna]

M[0][0] = 3;	M[0][1] = 8;	M[0][2] = 1;	M[0][3] = 5;
M[1][0] = 0;	M[1][1] = 2;	M[1][2] = 4;	M[1][3] = 7;
M[2][0] = 2;	M[2][1] = 5;	M[2][2] = 9;	M[2][3] = 3;

# Arranjos

## ■ Matriz (Exemplo)

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     const int NL = 3; // número de linhas
06     const int NC = 5; // número de colunas
07     const int K = 2; // fator para multiplicação
08     int M[NL][NC], I, J;
09     for (I = 0; I < NL; I = I + 1)
10         for (J = 0; J < NC; J = J + 1)
11         {
12             printf("Forneca o elemento da linha %d e coluna %d: ", I, J);
13             scanf("%d", &M[I][J]);
14         }
15     for (I = 0; I < NL; I = I + 1)
16         for (J = 0; J < NC; J = J + 1)
17         {
18             M[I][J] = M[I][J] * K;
19         }
20     printf("\nResultado:\n\n");
21     for (I = 0; I < NL; I = I + 1)
22     {
23         for (J = 0; J < NC; J = J + 1)
24             printf("%8d", M[I][J]);
25         printf("\n");
26     }
27     getch();
28 }
```

# Arranjos

## ■ Arranjos como argumentos de módulos

```
int somaElementosDeVetor(int vetor[ ], int colunas)
{
    int i, soma = 0;
    for (i = 0; i < colunas; i = i+1)
        soma = soma + vetor[i];
    return soma;
}
```



# Arranjos

## ■ Arranjos como argumentos de módulos

```
int somaElementosDeVetor(int vetor[ ], int colunas)
{
    int i, soma = 0;
    for (i = 0; i < colunas; i = i+1)
        soma = soma + vetor[i];
    return soma;
}
```

A quantidade de colchetes determina a dimensão do arranjo.

# Arranjos

## ■ Arranjos como argumentos de módulos

```
int somaElementosDeVetor(int vetor[ ], int colunas)
{
    int i, soma = 0;
    for (i = 0; i < colunas; i = i+1)
        soma = soma + vetor[i];
    return soma;
}
```

O primeiro par de colchetes fica vazio na implementação do módulo.

A quantidade de colchetes determina a dimensão do arranjo.

# Arranjos

## ■ Arranjos como argumentos de módulos

```
int somaElementosDeVetor(int vetor[ ], int colunas)
{
    int i, soma = 0;
    for (i = 0; i < colunas; i = i+1)
        soma = soma + vetor[i];
    return soma;
}
```

```
main()
{
    const int N = 4;
    int v[N], i;
    for (i = 0; i < N; i = i+1)
        v[i] = i;
    printf ("Soma dos elementos do vetor: %d", somaElementosDeVetor(v, N));
}
```

# Arranjos

## ■ Arranjos como argumentos de módulos

```
int somaElementosDeVetor(int vetor[ ], int colunas)
{
    int i, soma = 0;
    for (i = 0; i < colunas; i = i+1)
        soma = soma + vetor[i];
    return soma;
}
```

```
main()
{
    const int N = 4;
    int v[N], i;
    for (i = 0; i < N; i = i+1)
        v[i] = i;
    printf ("Soma dos elementos do vetor: %d", somaElementosDeVetor(v, N));
}
```

Os colchetes não aparecem na chamada ao módulo.

# Arranjos

## ■ Arranjos como argumentos de módulos

```
int somaElementosDeVetor(int vetor[ ], int colunas)
{
    int i, soma = 0;
    for (i = 0; i < colunas; i = i+1)
        soma = soma + vetor[i];
    return soma;
}
```

Em C, os arranjos, quando argumentos de módulos, são automaticamente passados por referência.

```
main()
{
    const int N = 4;
    int v[N], i;
    for (i = 0; i < N; i = i+1)
        v[i] = i;
    printf ("Soma dos elementos do vetor: %d", somaElementosDeVetor(v, N));
}
```

Os colchetes não aparecem na chamada ao módulo.

# Arranjos

## ■ Coleções implementadas com ponteiros

```
/******  
Vetores implementados com ponteiros  
*****/  
#include <stdio.h>  
#include <stdlib.h>  
main()  
{  
    int * vetor;  
    int i, j, I, J;  
    printf("Numero de colunas: ");  
    scanf("%d", &I);  
    vetor = (int *)malloc(sizeof(int[I]));  
    for(i = 0; i < I; i = i+1)  
    {  
        vetor[i] = i;  
        printf("%d", vetor[i]);  
    }  
    free(vetor);  
}
```

# Arranjos

## ■ Coleções implementadas com ponteiros

```
/******  
  Matrizes implementadas com ponteiros  
******/  
#include <stdio.h>  
#include <stdlib.h>  
main()  
{  
    int ** matriz;  
    int i, j, I, J;  
    printf("Numero de linhas : ");  
    scanf("%d", &I);  
    printf("Numero de colunas: ");  
    scanf("%d", &J);  
    matriz = (int **)malloc(sizeof(int*[I]));  
    for(i = 0; i < I; i = i+1)  
        matriz[i] = (int *)malloc(sizeof(int[J]));
```

```
    for(i = 0; i < I; i = i+1)  
    {  
        for(j = 0; j < J; j = j+1)  
        {  
            matriz[i][j] = i+j;  
            printf("%d", matriz[i][j]);  
        }  
        printf("\n");  
    }  
    for(i = 0; i < I; i = i+1)  
        free(matriz[i]);  
    free(matriz);  
}
```

# Arranjos

## ■ Coleções implementadas com ponteiros

```
/******  
  Tensores implementados com ponteiros  
******/  
#include <stdio.h>  
#include <stdlib.h>  
main()  
{  
    int *** tensor;  
    int i, j, k, I, J, K;  
    printf("Numero de paginas: ");  
    scanf("%d", &I);  
    printf("Numero de linhas : ");  
    scanf("%d", &J);  
    printf("Numero de colunas: ");  
    scanf("%d", &K);  
    tensor = (int ***)malloc(sizeof(int**[I]));  
    for(i = 0; i < I; i = i+1)  
    {  
        tensor[i] = (int **)malloc(sizeof(int*[J]));  
        for(j = 0; j < J; j = j+1)  
            tensor[i][j] = (int *)malloc(sizeof(int[K]));  
    }  
}
```



# Arranjos

## ■ Coleções implementadas com ponteiros

```
/******  
  Tensores implementados com ponteiros  
******/  
#include <stdio.h>  
#include <stdlib.h>  
main()  
{  
    int *** tensor;  
    int i, j, k, I, J, K;  
    printf("Numero de paginas: ");  
    scanf("%d", &I);  
    printf("Numero de linhas : ");  
    scanf("%d", &J);  
    printf("Numero de colunas: ");  
    scanf("%d", &K);  
    tensor = (int ***)malloc(sizeof(int **)*I);  
    for(i = 0; i < I; i = i+1)  
    {  
        tensor[i] = (int **)malloc(sizeof(int *)*J);  
        for(j = 0; j < J; j = j+1)  
            tensor[i][j] = (int *)malloc(sizeof(int)*K);  
        for(k = 0; k < K; k = k+1)  
        {  
            tensor[i][j][k] = i+j+k;  
            printf("%d", tensor[i][j][k]);  
        }  
        printf("\n");  
    }  
    printf("\n");  
    for(i = 0; i < I; i = i+1)  
    {  
        for(j = 0; j < J; j = j+1)  
            free(tensor[i][j]);  
        free(tensor[i]);  
    }  
    free(tensor);  
}
```

---

# Arranjos

- Exercícios do capítulo 6



---

# Manipulação de textos

## ■ Declaração

**char** nome[tamanho];

**char** \* nome;

---

# Manipulação de textos

## ■ Declaração

```
char nome[tamanho];  
char * nome;
```

## ■ Atribuição

```
strcpy(identificador, texto);
```

# Manipulação de textos

## ■ Declaração

```
char nome[tamanho];  
char * nome;
```

## ■ Atribuição

```
strcpy(identificador, texto);
```

## ■ Comparação

```
strcmp(texto1, texto2);
```

```
-1, se “texto1” < “texto2”;  
0, se “texto1” = “texto2”;  
+1, se “texto1” > “texto2”.
```

# Manipulação de textos

## ■ Declaração

```
char nome[tamanho];  
char * nome;
```

## ■ Atribuição

```
strcpy(identificador, texto);
```

## ■ Comparação

```
strcmp(texto1, texto2);
```

```
-1, se “texto1” < “texto2”;  
0, se “texto1” = “texto2”;  
+1, se “texto1” > “texto2”.
```

*strcpy* e *strcmp* são  
comandos da  
biblioteca *string.h*

---

# Manipulação de textos

## ■ Concatenação

**strcat**(identificador, texto);



# Manipulação de textos

- Concatenação

**strcat**(identificador, texto);

- Comprimento

**strlen**(texto);

# Manipulação de textos

## ■ Concatenação

**strcat**(identificador, texto);

## ■ Comprimento

**strlen**(texto);

*strcat* e *strlen* são comandos da biblioteca *string.h*

# Manipulação de textos

## ■ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 #include <string.h>
04 main()
05 {
06     char texto1[80], texto2[80];
07     int comprimento1, comprimento2, comparacao;
08     strcpy(texto1, "CEFET");
09     strcpy(texto2, texto1);
10     strcat(texto2, "/RJ");
11     comprimento1 = strlen(texto1);
12     comprimento2 = strlen(texto2);
13     comparacao = strcmp(texto1, texto2);
14     printf("O primeiro texto e %s.\n", texto1);
15     printf("O segundo texto e %s.\n", texto2);
16     printf("O comprimento do primeiro texto e %d.\n", comprimento1);
17     printf("O comprimento do segundo texto e %d.\n", comprimento2);
```

# Manipulação de textos

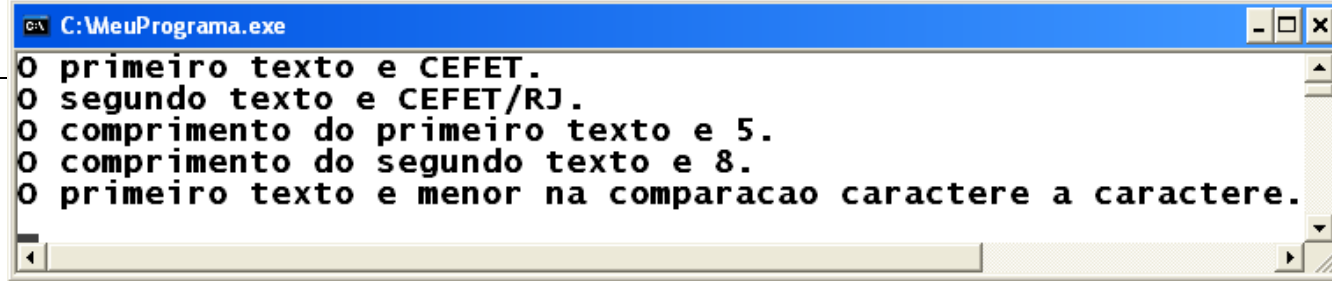
## ■ Exemplo

```
18  if (comparacao == 0)
19      printf("Os dois textos sao identicos.\n");
20  else if (comparacao == 1)
21      printf("O primeiro texto e maior na comparacao caractere
           a caractere.\n");
22      else printf("O primeiro texto e menor na comparacao
           caractere a caractere.\n");
23  getch();
24 }
```

# Manipulação de textos

## ■ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 #include <string.h>
04 main()
05 {
06     char texto1[80], texto2[80];
07     int comprimento1, comprimento2, comparacao;
08     strcpy(texto1, "CEFET");
09     strcpy(texto2, texto1);
10     strcat(texto2, "/RJ");
11     comprimento1 = strlen(texto1);
12     comprimento2 = strlen(texto2);
13     comparacao = strcmp(texto1, texto2);
14     printf("O primeiro texto e %s.\n", texto1);
15     printf("O segundo texto e %s.\n", texto2);
16     printf("O comprimento do primeiro texto e %d.\n", comprimento1);
17     printf("O comprimento do segundo texto e %d.\n", comprimento2);
18     if (comparacao == 0)
19         printf("Os dois textos sao identicos.\n");
20     else if (comparacao == 1)
21         printf("O primeiro texto e maior na comparacao caractere a caractere.\n");
22     else printf("O primeiro texto e menor na comparacao caractere a caractere.\n");
23     getch();
24 }
```



```
C:\MeuPrograma.exe
O primeiro texto e CEFET.
O segundo texto e CEFET/RJ.
O comprimento do primeiro texto e 5.
O comprimento do segundo texto e 8.
O primeiro texto e menor na comparacao caractere a caractere.
```

# Manipulação de textos

## ■ Funções para caracteres

- ❑ **toupper**(caractere);
- ❑ **tolower**(caractere);

*tolower e toupper  
estão definidas no  
arquivo de cabeçalho  
ctype.h.*

- ❑ *Em C, se atribuirmos um caractere a uma variável inteira, esta variável passará a conter o valor ASCII deste caractere. Como contraposto, se atribuirmos um número inteiro de 0 a 255 a uma variável caractere, esta variável passará a conter o caractere associado ao valor ASCII deste código.*

# Manipulação de textos

## ■ Exemplo

```
01 #include <stdio.h>
02 #include <conio.h>
03 #include <string.h>
04 #include <ctype.h>
05 main()
06 {
07     int i;
08     char entrada[80], saida[80];
09     printf("Escreva um texto qualquer: ");
10     gets(entrada);
11     i = 0;
12     while (i <= strlen(entrada))
13     {
14         saida[i] = toupper(entrada[i]); //saida[i] = tolower(entrada[i]);
15         i = i + 1;
16     }
17     printf("%s", saida);
18     getch();
19 }
```

# Manipulação de textos

## ■ Exemplos

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int inteiro;
06     inteiro = 'A';
07     printf("%d", inteiro);
08     getch();
09 }
```

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     char caractere;
06     caractere = 122;
07     printf("%c", caractere);
08     getch();
09 }
```



# Manipulação de textos

## ■ Exemplos

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     int inteiro;
06     inteiro = 'A';
07     printf("%d", inteiro);
08     getch();
09 }
```

```
01 #include <stdio.h>
02 #include <conio.h>
03 main()
04 {
05     char caractere;
06     caractere = 122;
07     printf("%c", caractere);
08     getch();
09 }
```

O resultado de cada programa será, respectivamente:

- escrever na tela o inteiro 65, já que 65 é o código ASCII do caractere 'A';
- escrever na tela o caractere 'z', uma vez que 'z' é o caractere cujo código ASCII é dado por 122.

---

# Manipulação de textos

- Exercícios do capítulo 7



---

# Tipos definidos pelo programador

- Tipos primitivos

- char

- int

- float

# Tipos definidos pelo programador

## ■ Tipos primitivos

- char

- int

- float

- ponteiros

- arranjos

---

# Tipos definidos pelo programador

## ■ Tipos primitivos

- ❑ char
- ❑ int
- ❑ float
- ❑ ponteiros
- ❑ arranjos

## ■ Outros tipos

- ❑ Tipos enumerados
  - ❑ Tipos de registro
-

---

# Tipos definidos pelo programador

- Tipos enumerados

- Enumera os valores do novo tipo
- Sintaxe

```
enum tipoEnumerado { descricao1, descricao2, ..., descricaoN };
```

# Tipos definidos pelo programador

## ■ Tipos enumerados

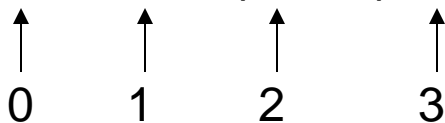
- Enumera os valores do novo tipo
- Sintaxe

```
enum tipoEnumerado { descricao1, descricao2, ..., descricaoN };
```

## □ Exemplos:

a) **enum** TNaipes {paus, ouros, copas, espadas};

valores sinônimos:    0        1        2        3





# Tipos definidos pelo programador

## ■ Tipos enumerados


- Enumera os valores do novo tipo
- Sintaxe

```
enum tipoEnumerado { descricao1, descricao2, ..., descricaoN };
```

### □ Exemplos:


a) **enum** TNaipes {paus, ouros, copas, espadas};

valores sinônimos:    0        1        2        3



b) **enum** TDiaDaSemana {domingo=1, segunda, terca, quarta, quinta, sexta, sabado};

valores sinônimos:    1        2        3        4        5        6        7



# Tipos definidos pelo programador

## ■ Tipos enumerados


- Enumera os valores do novo tipo
- Sintaxe

```
enum tipoEnumerado { descricao1, descricao2, ..., descricaoN };
```

### □ Exemplos:


a) **enum** TNaipes {paus, ouros, copas, espadas};

valores sinônimos:    0        1        2        3



b) **enum** TDiaDaSemana {domingo=1, segunda, terca, quarta, quinta, sexta, sabado};

valores sinônimos:    1        2        3        4        5        6        7



Operadores lógicos são válidos para as descrições de um tipo enumerado.

---

# Tipos definidos pelo programador

- Tipos enumerados
  - Declaração
    - Junto com a declaração do tipo
  - Depois da declaração do tipo

# Tipos definidos pelo programador

- Tipos enumerados

- Declaração

- Junto com a declaração do tipo

**enum** tipoEnumerado {valor1, valor2, ...} identificador1 [, identificador2, ..., identificadorN];

- Depois da declaração do tipo

# Tipos definidos pelo programador

- Tipos enumerados

- Declaração

- Junto com a declaração do tipo

**enum** tipoEnumerado {valor1, valor2, ...} identificador1 [, identificador2, ..., identificadorN];

**enum** TNaipes {paus, ouros, espadas, copas} naipesDaMesa, naipesDoJogador;

- Depois da declaração do tipo

# Tipos definidos pelo programador

- Tipos enumerados

- Declaração

- Junto com a declaração do tipo

```
enum tipoEnumerado {valor1, valor2, ...} identificador1 [, identificador2, ..., identificadorN];
```

```
enum TNaipes {paus, ouros, espadas, copas} naipesDaMesa, naipesDoJogador;
```

- Depois da declaração do tipo

```
enum tipoEnumerado identificador1 [, identificador2, ..., identificadorN];
```

# Tipos definidos pelo programador

- Tipos enumerados

- Declaração

- Junto com a declaração do tipo

```
enum tipoEnumerado {valor1, valor2, ...} identificador1 [, identificador2, ..., identificadorN];
```

```
enum TNaipes {paus, ouros, espadas, copas} naipesDaMesa, naipesDoJogador;
```

- Depois da declaração do tipo

```
enum tipoEnumerado identificador1 [, identificador2, ..., identificadorN];
```

```
enum TNaipes naipesDaMesa, naipesDoJogador;
```

---

---

# Tipos definidos pelo programador

- Tipos de registro
  - Registram mais de um componente



# Tipos definidos pelo programador

## ■ Tipos de registro

- Registram mais de um componente

- Sintaxe

```
struct nomeDoRegistro
{
    tipo1 nomeDoCampo1_1 [, nomeDoCampo1_2, ..., nomeDoCampo1_W];
    tipo2 nomeDoCampo2_1 [, nomeDoCampo2_2, ..., nomeDoCampo2_X];
    ...
    tipoN nomeDoCampoN_1 [, nomeDoCampoN_2, ..., nomeDoCampoN_Y];
};
```

# Tipos definidos pelo programador

## ■ Tipos de registro

- Registram mais de um componente

- Sintaxe

```
struct nomeDoRegistro
{
    tipo1 nomeDoCampo1_1 [, nomeDoCampo1_2, ..., nomeDoCampo1_W];
    tipo2 nomeDoCampo2_1 [, nomeDoCampo2_2, ..., nomeDoCampo2_X];
    ...
    tipoN nomeDoCampoN_1 [, nomeDoCampoN_2, ..., nomeDoCampoN_Y];
};
```

- Exemplos

```
struct TData
{
    int dia;
    char * mes;
    int ano;
};
```

```
struct TData
{
    int dia, ano;
    char * mes;
};
```

---

# Tipos definidos pelo programador

- Tipos de registro
    - Declaração
      - Junto com a declaração do tipo
    - Depois da declaração do tipo
-

# Tipos definidos pelo programador

- Tipos de registro

- Declaração

- Junto com a declaração do tipo

**struct** tipoDeRegistro {...} identificador1 [, identificador2, ..., identificadorN];

- Depois da declaração do tipo

# Tipos definidos pelo programador

- Tipos de registro

- Declaração

- Junto com a declaração do tipo

```
struct tipoDeRegistro {...} identificador1 [, identificador2, ..., identificadorN];
```

```
struct TData
```

```
{
```

```
    int dia, ano;
```

```
    char mes[10];
```

```
} hoje, nascimento;
```

- Depois da declaração do tipo

# Tipos definidos pelo programador

- Tipos de registro

- Declaração

- Junto com a declaração do tipo

```
struct tipoDeRegistro {...} identificador1 [, identificador2, ..., identificadorN];
```

```
struct TData
```

```
{
```

```
    int dia, ano;
```

```
    char mes[10];
```

```
} hoje, nascimento;
```

- Depois da declaração do tipo

```
struct tipoDeRegistro identificador1 [, identificador2, ..., identificadorN];
```

# Tipos definidos pelo programador

## ■ Tipos de registro

### □ Declaração

#### ■ Junto com a declaração do tipo

```
struct tipoDeRegistro {...} identificador1 [, identificador2, ..., identificadorN];
```

```
struct TData
```

```
{
```

```
    int dia, ano;
```

```
    char mes[10];
```

```
} hoje, nascimento;
```

#### ■ Depois da declaração do tipo

```
struct tipoDeRegistro identificador1 [, identificador2, ..., identificadorN];
```

```
struct TData hoje, nascimento;
```

---

# Tipos definidos pelo programador

- Tipos de registro
  - Manipulação do registro inteiro
    - `struct1 = struct2;`



---

# Tipos definidos pelo programador

- Tipos de registro
  - Manipulação do registro inteiro
    - `struct1 = struct2;`

`nascimento = hoje;`

---

# Tipos definidos pelo programador

- Tipos de registro

- Manipulação do registro inteiro

- `struct1 = struct2;`

`nascimento = hoje;`

- Manipulação de um campo do registro

- Designador de campo `'.'`

# Tipos definidos pelo programador

- Tipos de registro

- Manipulação do registro inteiro

- `struct1 = struct2;`

`nascimento = hoje;`

- Manipulação de um campo do registro

- Designador de campo `'.'`

`nomeDaVariável.nomeDoCampo`

```
scanf("%d", &hoje.ano);  
nascimento.ano = hoje.ano;  
printf("%d", nascimento.ano);
```

---

# Tipos definidos pelo programador

- Tipos de registro
  - Estruturas mistas
    - O campo de um tipo de registro é outro tipo de registro

# Tipos definidos pelo programador

- Tipos de registro

- Estruturas mistas

- O campo de um tipo de registro é outro tipo de registro

- Exemplo

```
struct TData  
{  
    int dia, mes, ano;  
};
```

```
struct TPessoa  
{  
    char * nome;  
    struct TData dataDeNascimento;  
};  
struct TPessoa pessoa;
```

# Tipos definidos pelo programador

## ■ Tipos de registro

### □ Estruturas mistas

- O campo de um tipo de registro é outro tipo de registro

### □ Exemplo

```
struct TData
{
    int dia, mes, ano;
};
```

```
struct TPessoa
{
    char * nome;
    struct TData dataDeNascimento;
};
struct TPessoa pessoa;
```

### □ Acesso

```
scanf("%d", &pessoa.dataDeNascimento.dia);
```

# Tipos definidos pelo programador

## ■ Tipos de registro

### □ Campos ponteiros / ponteiros para structs

```
main()
{
    struct TPessoa
    {
        char * nome;
        struct TData
        {
            int dia;
            int mes;
            int ano;
        } nascimento;
    } * pessoa;
    ...
}
```

# Tipos definidos pelo programador

## ■ Tipos de registro

### □ Campos ponteiros / ponteiros para structs

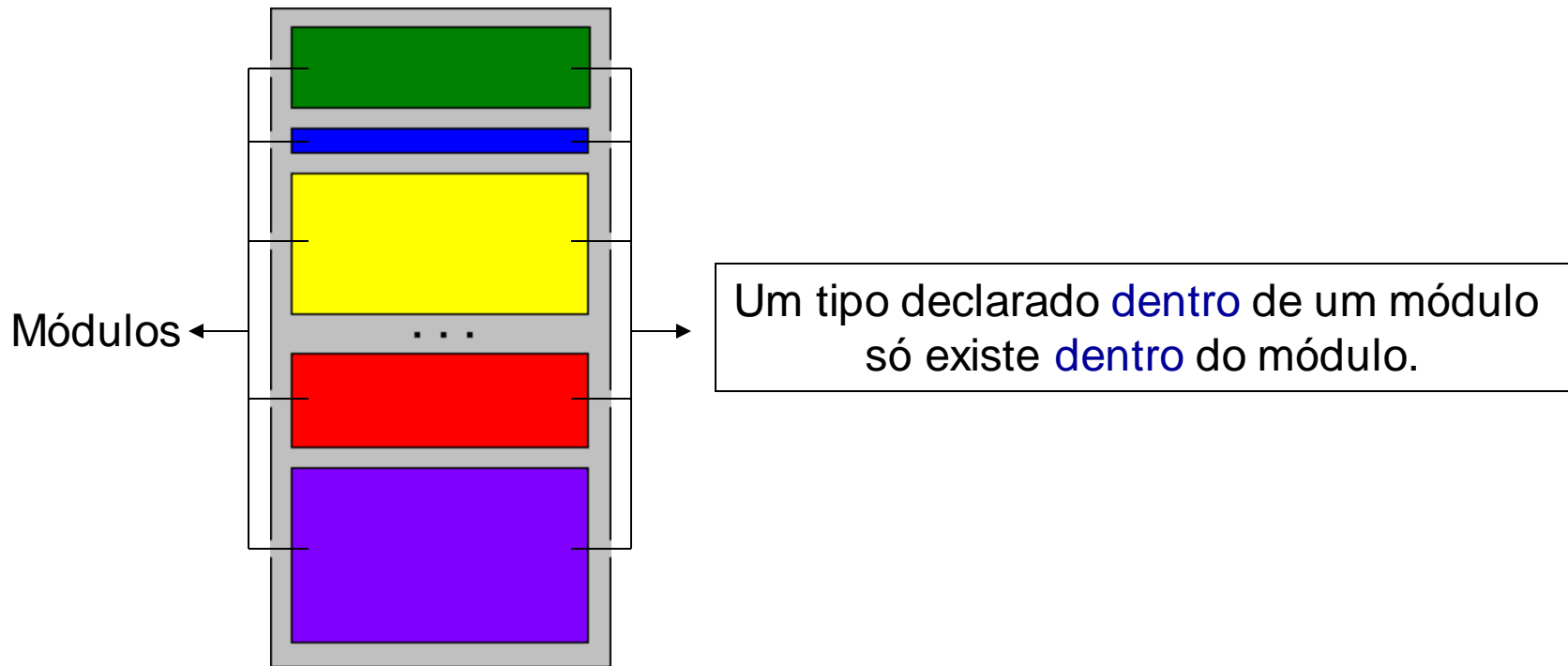
```
main()
{
    struct TPessoa
    {
        char * nome;
        struct TData
        {
            int dia;
            int mes;
            int ano;
        } nascimento;
    } * pessoa;
    ...
}
```

```
    pessoa = (struct TPessoa *)malloc(sizeof(struct TPessoa));
    pessoa->nome = (char*)malloc(sizeof(char));
    //(*pessoa).nome = (char*)malloc(sizeof(char));
    ...
    printf ("NOME: ");
    gets(pessoa->nome);
    //gets ((*pessoa).nome);
    ...
    printf ("%s",pessoa->nome);
    //printf ("%s", (*pessoa).nome);
    ...
    free(pessoa->nome);
    free(pessoa);
}
```



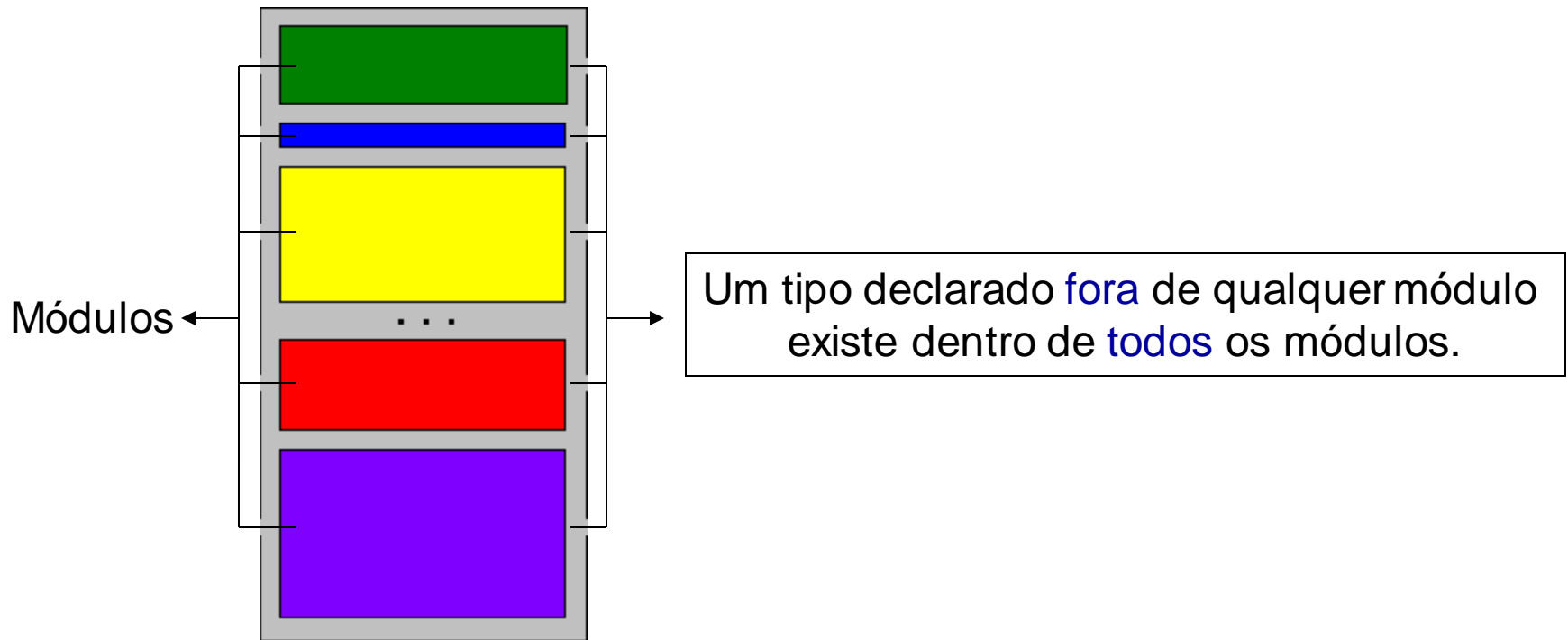
# Tipos definidos pelo programador

- Estruturas locais X estruturas globais



# Tipos definidos pelo programador

- Estruturas locais X estruturas globais



# Tipos definidos pelo programador

## ■ Exemplo completo

```
01 #include <stdio.h>
02 #include <conio.h>

03 enum TMes {Janeiro=1, Fevereiro, Marco, Abril,
              Maio, Junho, Julho, Agosto, Setembro,
              Outubro, Novembro, Dezembro};

04 struct TData
05 {
06     int dia;
07     enum TMes mes;
08     int ano;
09 };

10 int AnoBissexto (int ano)
11 {
12     if (ano % 4 == 0 && ano % 100 != 0 || ano % 400 == 0)
13         return(1);
14     else return(0);
15 }
```

```
16 char * TMesParaTexto (enum TMes mes)
17 {
18     switch (mes)
19     {
20         case Janeiro : return("Janeiro");
21                         break;
22         case Fevereiro: return("Fevereiro");
23                         break;
24         case Marco    : return("Marco");
25                         break;
26         ...
42         case Dezembro : return("Dezembro");
43                         break;
44     }
45 }
```

# Tipos definidos pelo programador

## ■ Exemplo completo

```
46 int ValidaData (struct TData data)
47 {
48     switch (data.mes)
49     {
50         case Janeiro:
51         case Marco:
52         case Maio:
53         case Julho:
54         case Agosto:
55         case Outubro:
56         case Dezembro: if (data.dia <= 31)
57                         return(1);
58                         else return(0);
59                         break;
```

```
60     case Abril:
61     case Junho:
62     case Setembro:
63     case Novembro: if (data.dia <= 30)
64                     return(1);
65                     else return(0);
66                     break;
67     case Fevereiro: if (AnoBissexto(data.ano))
68                     {
69                         if (data.dia <= 29)
70                             return(1);
71                         else return(0);
72                     }
73     else {
74         if (data.dia <= 28)
75             return(1);
76         else return(0);
77     }
78     default : return(0);
79 }
80 }
```

# Tipos definidos pelo programador

## ■ Exemplo completo

```
81 main()
82 {
83     struct TPessoa
84     {
85         char nome[80];
86         struct TData dataDeNascimento;
87     };
88     struct TPessoa pessoa;
89     printf("Informe seu nome: ");
90     gets(pessoa.nome);
91     printf("%s, informe sua data de nascimento no formato dd/mm/aaaa: ", pessoa.nome);
92     scanf("%d/%d/%d", &pessoa.dataDeNascimento.dia, &pessoa.dataDeNascimento.mes,
            &pessoa.dataDeNascimento.ano);
93     if (ValidaData(pessoa.dataDeNascimento))
94         printf("Voce nasceu em %d de %s de %d.", pessoa.dataDeNascimento.dia,
                TMesParaTexto(pessoa.dataDeNascimento.mes), pessoa.dataDeNascimento.ano);
95     else printf("A data fornecida e invalida. O programa sera fechado.");
96     getch();
97 }
```

# Tipos definidos pelo programador

- O uso de *typedef*
  - Cria um sinônimo para um tipo de dado já existente

```
typedef int inteiro;
```

```
typedef enum _TMes_ {janeiro, fevereiro, ..., dezembro} TMes;
```

```
typedef struct _TData_  
{  
    inteiro dia, ano;  
    TMes mes;  
} TData;
```

```
main()  
{  
    TData data;  
    ...  
}
```

---

# Tipos definidos pelo programador

- Exercícios do capítulo 8





# Arquivos

- Meio não volátil
  - Dados não são perdidos ao término do programa
- Tipos de arquivos em C
  - Arquivos de registros tipados
  - Arquivos de texto
- Declaração de um arquivo

*FILE* é uma estrutura presente no arquivo de cabeçalho *stdlib.h*

```
FILE * nomeInternoDoArquivo;
```

# Arquivos

## ■ O comando *fopen*

```
arquivoInterno = fopen(arquivoExterno, modo);
```

```
arquivo = fopen ("c:\alunos.dat", "wb+");
```

Modo	Significado
r	Abre um arquivo texto para leitura
w	Cria um arquivo texto para escrita
a	Anexa a um arquivo texto
r+	Abre um arquivo texto para leitura / escrita
w+	Cria um arquivo texto para leitura / escrita
a+	Anexa ou cria um arquivo texto para leitura / escrita
rb	Abre um arquivo binário para leitura
wb	Cria um arquivo binário para escrita
ab	Anexa a um arquivo binário
rb+	Abre um arquivo binário para leitura / escrita
wb+	Cria um arquivo binário para leitura / escrita
ab+	Anexa ou cria um arquivo binário para leitura / escrita

# Arquivos

- O comando *rewind*

```
rewind(arquivoInterno);  
rewind(arquivo);
```

- O comando *feof*

```
feof(arquivoInterno);  
if (feof(arquivo)) break;
```

# Arquivos

- O comando *fclose*

```
fclose(arquivoInterno);
```

```
fclose(arquivo);
```

- O comando *remove*

```
remove(arquivoExterno);
```

```
remove("c:\alunos.dat");
```

# Arquivos

Para serem usados em arquivos tipados.

## ■ O comando *fwrite*

```
fwrite(endereco, tamanhoDeCadaRegistro, quantidadeDeRegistros,  
       arquivoInterno);
```

```
fwrite(&aluno, sizeof(struct TAluno), 1, arquivo);
```

## ■ O comando *fread*

```
fread(endereco, tamanhoDeCadaRegistro, quantidadeDeRegistros,  
      arquivoInterno);
```

```
fread(&aluno, sizeof(struct TAluno), 1, arquivo);
```

# Arquivos

Para ser usado em arquivos tipados.

## ■ O comando *fseek*

```
fseek(arquivoInterno, deslocamento, origem);
```

```
fseek(arquivo, registros * sizeof(struct TAluno), 0);
```

Move a posição corrente de um arquivo para um deslocamento a partir de uma origem que pode ser:

- 0 – começo do arquivo
- 1 – posição corrente do arquivo
- 2 – fim do arquivo

# Arquivos

Para serem usados em arquivos texto.

- Os comandos *fprintf* e *fscanf*
  - Análogos a *printf* e *scanf*
  - Inclusão de um parâmetro inicial (arquivo interno)

```
fprintf(arquivo, "%3d %5d\n", i, i * i);
```

```
fscanf(arquivo, "%3d %5d", &numero, &quadrado);
```

# Arquivos

Para ser usado em arquivos texto.

## ■ O comando *fgets*

```
fgets(identificador, tamanhoEmBytes, arquivoInterno);
```

```
fgets(texto, 80, arquivo);
```

A leitura destes bytes é imediatamente encerrada se encontrado um caractere de quebra de linha.



---

# Arquivos

- Exercícios do capítulo 9

