

# Introdução à C/C++ para programadores Python

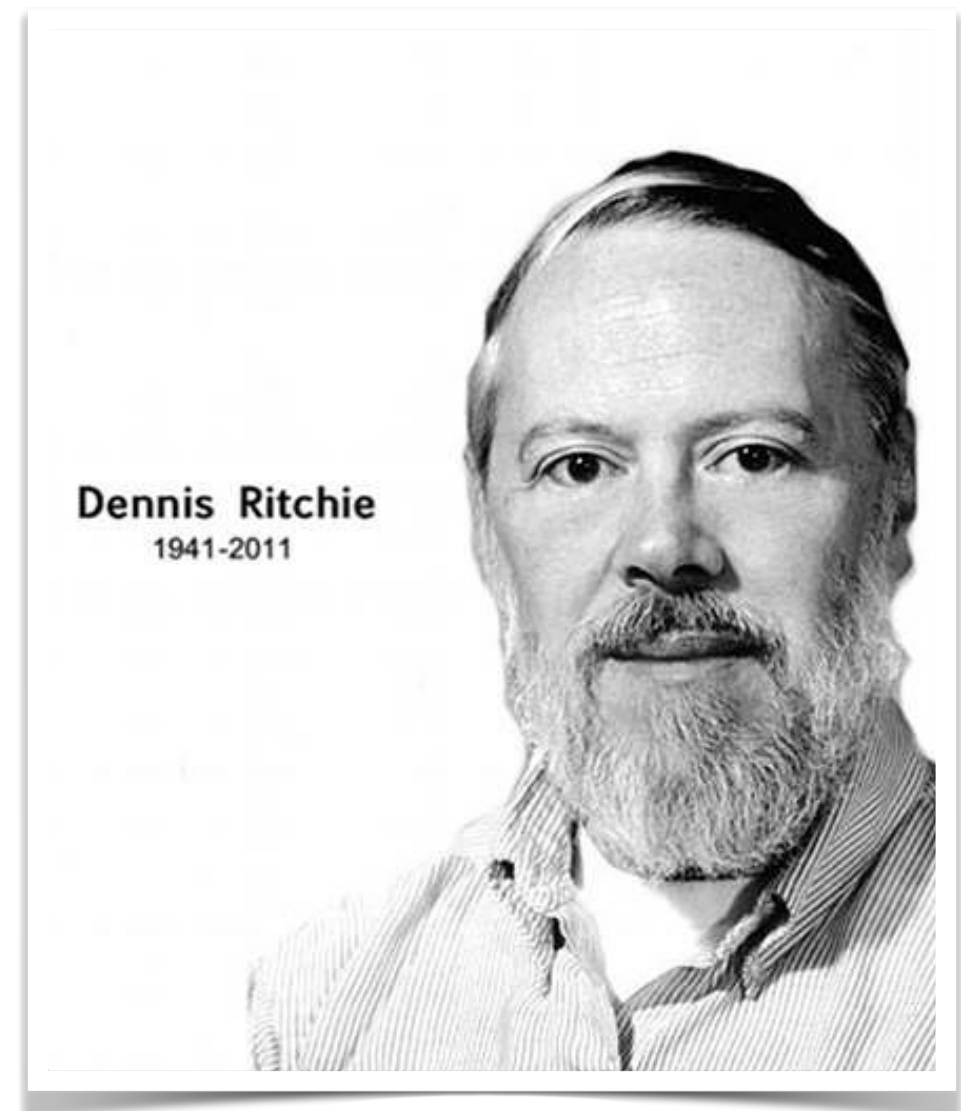
Algoritmos e Programação 2

Prof. Dr. Anderson Bessa da Costa

Universidade Federal de Mato Grosso do Sul

# História C/C++

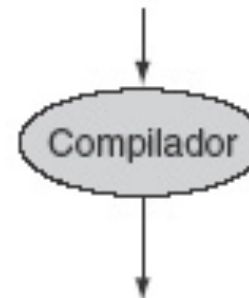
- A linguagem C foi criada inicialmente por Dennis M. Ritchie e Ken Thompson no laboratório Bell em 1972
- Baseada na linguagem B, de Thompson, evolução da antiga linguagem BCPL
- C++ é uma evolução de C (~80s)



# Abstração

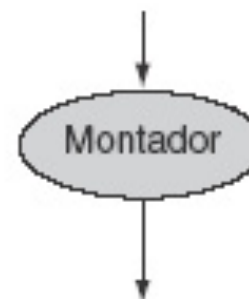
Programa em  
linguagem de  
alto nível  
(em C)

```
swap(int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```



Programa em  
assembly  
(para MIPS)

```
swap:
    muli $2, $5, 4
    add $2, $4, $2
    lw $15, 0($2)
    lw $16, 4($2)
    sw $16, 0($2)
    sw $15, 4($2)
    jr $31
```



Programa  
binário em  
linguagem de  
máquina  
(para MIPS)

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
100011001111001000000000000000100
10101100111100100000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

# Primeiro Programa

```
1. // Um primeiro programa em C++
2. #include <stdio.h>
3.
4. int main() {
5.
6.     printf("Ola mundo!\n");
7.
8.     return 0;
9. }
```

```
% g++ hello_world.cpp
% ./a.out
Ola mundo!
%
```

# Principais Códigos Especiais em C/C++

<b>Códigos Especiais</b>	<b>Significado</b>
<code>\n</code>	Nova linha.
<code>\t</code>	Tabulação.
<code>\\</code>	\ - Barra invertida.
<code>\0</code>	Zero.
<code>\'</code>	Aspas simples (apóstrofo).
<code>\"</code>	Aspas dupla.

# Outro Programa: Adicione dois inteiros

```
// Programa de adicao
#include <stdio.h>

int main() {
    int a, b, soma; // declaracao

    printf("Entre com um inteiro\n"); // imprima
    scanf("%d", &a); // leia um inteiro

    printf("Entre com outro inteiro\n"); // imprima
    scanf("%d", &b); // leia um inteiro

    soma = a + b; // atribuicao de soma
    printf("A soma eh %d\n", soma); // imprima soma

    return 0; // indica que o programa finalizou com sucesso
}
```

```
Entre com um inteiro
50
Entre com outro inteiro
5
A soma eh 55
```

# Códigos de Formatação Printf

Códigos de formatação para printf()	Significado
%c	Caractere simples.
%d	Inteiro decimal com sinal.
%i	Inteiro decimal com sinal.
%e	Notação científica (e minúsculo).
%E	Notação científica (E maiúsculo).
%f	Ponto flutuante em decimal.
%g	Usa %e ou %f, o que for menor.
%G	Usa %E ou %f, o que for menor.
%o	Inteiro octal sem sinal.
%s	String de caracteres.
%u	Inteiro decimal sem sinal.
%x	Inteiro hexadecimal sem sinal (letras minúsculas).
%X	Inteiro hexadecimal sem sinal (letras maiúsculas).
%p	Ponteiro (endereço).
%n	Ponteiro inteiro.
%%	Imprime o caractere %.

# Scanf

- É o complemento de **printf()**
- Permite ler dados formatados da entrada padrão (teclado)
- Sua sintaxe é similar à de **printf()**
- A principal diferença está na lista de argumentos, que devem ser endereços de variáveis
- As definições necessárias ao uso de **scanf()** estão no arquivo **stdio.h**



# Exemplo 1: Múltiplas leituras scanf()

```
#include <stdio.h>

int main () {
    float p1, p2, p3, p4;
    double media;

    printf("Digite as notas das 4 provas: ");
    scanf("%f%f%f%f", &p1, &p2, &p3, &p4);
    media = (p1 + p2 + p3 + p4)/4;
    printf("MEDIA: %.2f\n", media);

    return 0;
}
```

# Códigos Formatação scanf()

Códigos de formatação para scanf()	Significado
%c	Caractere simples.
%d	Inteiro decimal com sinal.
%i	Inteiro decimal, hexadecimal ou octal.
%e	Notação científica.
%f	Ponto flutuante em decimal.
%g	Usa %e ou %f, o que for menor.
%o	Inteiro octal.
%s	String de caracteres.
%u	Inteiro decimal sem sinal.
%x	Inteiro hexadecimal.
%ld	Inteiro decimal longo.
%lf	Ponto flutuante longo (double).
%LF	Double longo.

Códigos formatação para scanf().

# Operador de Endereços (&)

- Toda variável ocupa uma certa localização na memória, e o seu endereço é o do primeiro *byte* ocupado por ela
- Um endereço é a referência que o computador usa para localizar as variáveis
- O operador **&** opera sobre o nome de uma variável e resulta o seu endereço

# Exemplo 2: Endereço da variável

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

```
int main () {  
    int n;
```

```
    n = 2;
```

```
    printf("Valor=%d, endereco=%p\n", n, &n);
```

```
    return 0;
```

```
}
```

Valor=2, endereco=0x16f0373e8

# Variáveis

- Aspecto fundamental de qualquer linguagem
- É um espaço de memória reservado para armazenar um certo **tipo** de dado e tendo um **nome** para referenciar o seu conteúdo
- Toda variável deve ser declarada antes de ser usada

# Tipos de Variáveis

- C++ adiciona o tipo **bool** (*true*, *false*) aos tipos primitivos de C

Tipo	Bits	Bytes	Escala
char	8	1	128 a 127
int	32	4	-2.147.483.648 a 2.147.483.647 (ambientes de 32 bits)
float	32	4	$3,4 \times 10^{-38}$ a $3,4 \times 10^{38}$
double	64	8	$1,7 \times 10^{-308}$ a $1,7 \times 10^{308}$
void	0	0	nenhum valor

Tipos de variáveis em C.

# Operadores Aritméticos

- C oferece os seguintes operadores aritméticos:
  - **Binários:** soma (+), subtração (-), multiplicação (\*), divisão (/) e módulo (%)
  - **Unário:** menos unário (-)

# Operadores Aritméticos de Atribuição

- Combinam operações aritméticas com a operação de atribuição

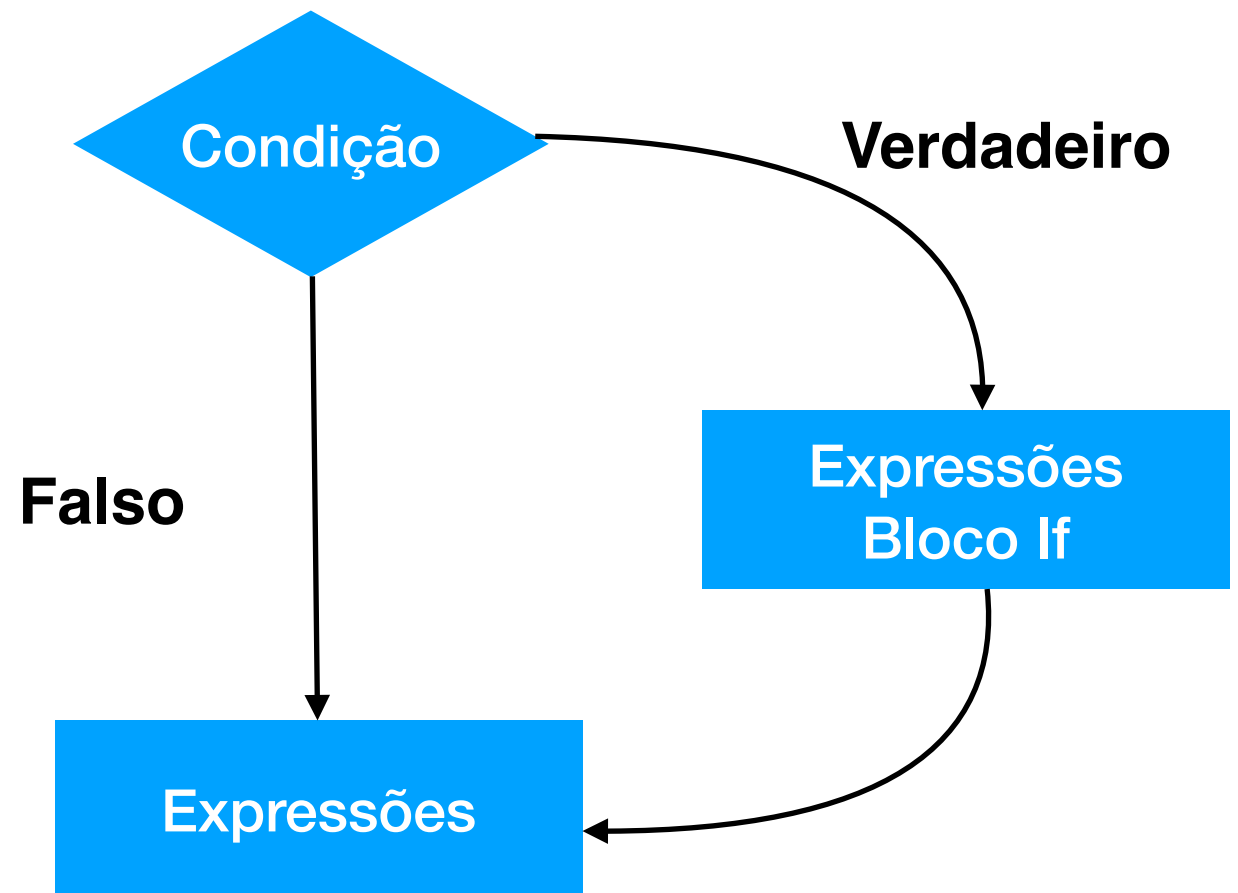
Operador	Exemplo	Comentário
<code>+=</code>	<code>x += y</code>	Equivale a <code>x = x + y.</code>
<code>-=</code>	<code>x -= y</code>	Equivale a <code>x = x - y.</code>
<code>*=</code>	<code>x *= y</code>	Equivale a <code>x = x * y.</code>
<code>/=</code>	<code>x /= y</code>	Equivale a <code>x = x / y.</code>
<code>%=</code>	<code>x %= y</code>	Equivale a <code>x = x % y.</code>



# Comandos de Decisão

# Desvio Condicional Simples

```
int main() {  
    if (<condição>) {  
  
    }  
  
    return 0;  
}
```



Comandos dentro do bloco **if** são executados caso a condição seja **VERDADEIRA**

# Desvio Condicional Simples (cont.)

- A **condição** em estruturas **if** podem ser formadas utilizando os operadores de igualdade e operadores relacionais

# Operadores Relacionais

- Realizam comparações
- O resultado obtido de uma relação é sempre um valor lógico (verdadeiro ou falso)

Operador	Exemplo	Comentário
==	x == y	O conteúdo de x é igual ao conteúdo de y.
!=	x != y	O conteúdo de x é diferente do conteúdo de y.
<=	x <= y	O conteúdo de x é menor ou igual ao conteúdo de y.
>=	x >= y	O conteúdo de x é maior ou igual ao conteúdo de y.
<	x < y	O conteúdo de x é menor que o conteúdo de y.
>	x > y	O conteúdo de x é maior que o conteúdo de y.

Operadores relacionais.

# Programa: Aprovado?

```
#include <stdio.h>

int main() {
    float media_final;

    printf("Qual a sua media final da disciplina?\n");
    scanf("%f", &media_final);

    // Toma uma decisão se nota maior ou igual a 6.0
    if (media_final >= 6.0) {
        printf("Aprovado!\n");
    }

    return 0;
}
```

```
Qual a sua media final da disciplina?
7
Aprovado!
Qual a sua media final da disciplina?
5
```

# Operadores Lógicos

- Também fazem comparações
- Operandos de operadores lógicos são avaliados como lógicos (0 ou 1), e não como quantidades numéricas


Operador	Operação
&&	Lógico E
	Lógico OU
!	Lógico NÃO


# Desvio Condicional Simples

```
if (0) {  
    <expressao1>;  
    <expressao2>;  
    <expressao3>;  
}
```

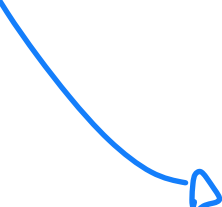
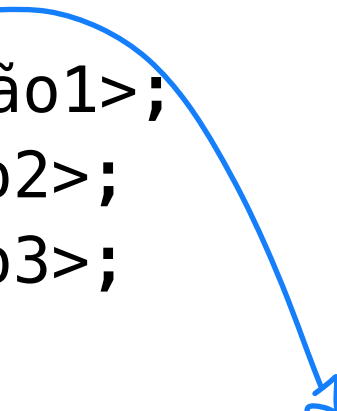
Nesse caso, nenhuma das 3 expressões dentro do **if** serão executadas.

# Desvio Condicional Simples (cont.)

```
if (0)   
    <expressão1>;  
    <expressão2>;  
    <expressão3>;
```



Nesse caso, serão executadas as expressões **expressão2** e **expressão3**, pois somente a **expressão1** pertence ao **if**.



**Podemos retirar as chaves que indicam a abertura e fechamento do bloco.**



# Desvio Condicional Simples em Sequência

```
int main() {  
    if (<condição1>) {  
  
    }  
    if (<condição2>) {  
  
    }  
    if (<condição3>) {  
  
    }  
  
    return 0;  
}
```

Nesse caso, cada um dos **if** é avaliado de forma independente.

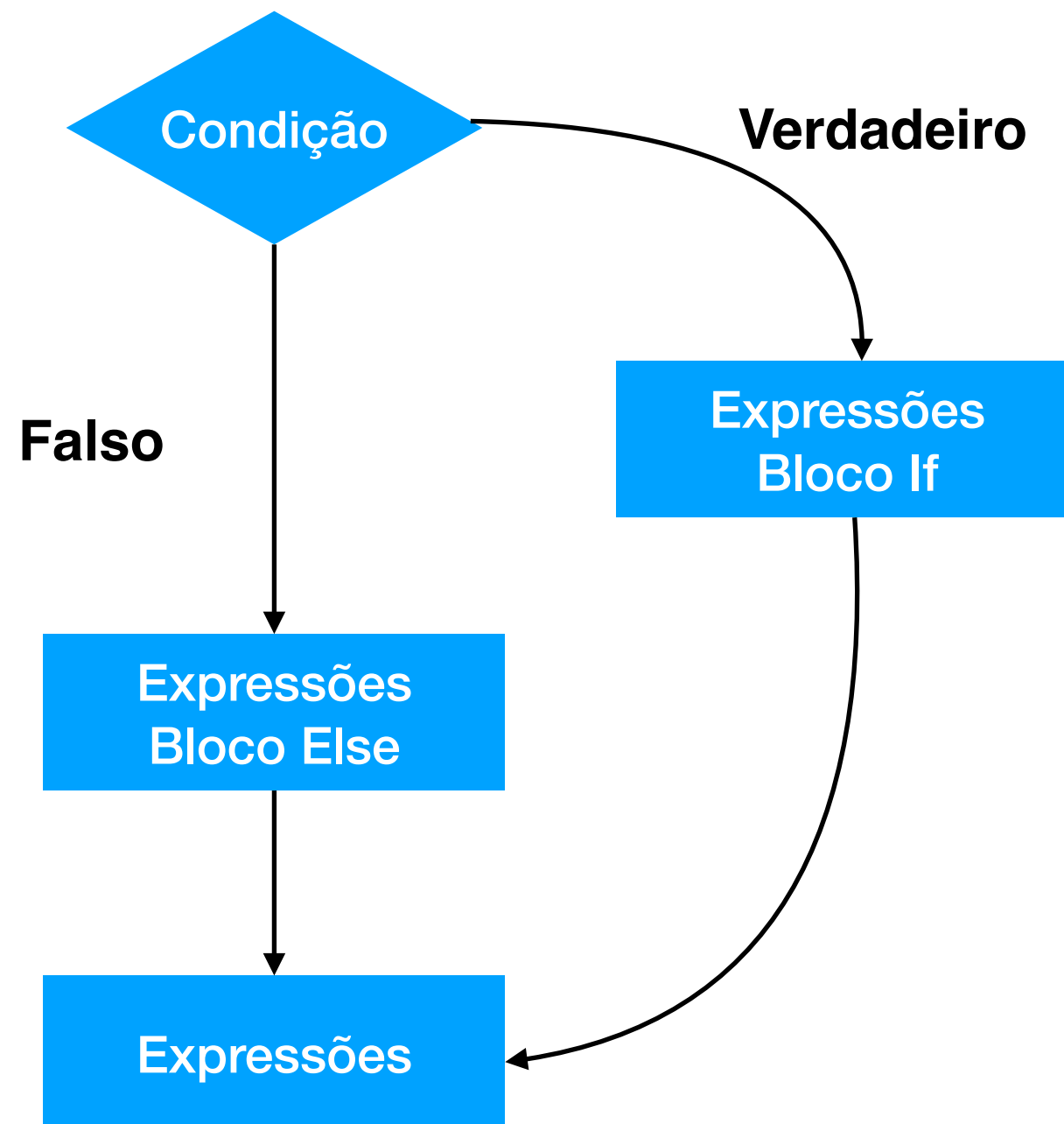
# Desvio Condicional Composto

```
int main() {  
    if (<condição>) {  
  
    }  
    else {  
  
    }  
  
    return 0;  
}
```

Comandos dentro do bloco **if** são executados caso a condição seja **VERDADEIRA**

Comandos dentro do bloco **else** são executados caso a condição seja **FALSA**

# Visualização Desvio Composto



# Desvio Condicional Composto (cont.)

```
int main() {  
  
    if (<condição1>) {  
  
    }  
    else if (<condição2>) {  
  
    }  
    else if (<condição3>) {  
  
    }  
    else {  
  
    }  
  
    return 0;  
}
```

O bloco **else if** é opcional, podendo não existir ou existir vários.

Em um desvio condicional composto, apenas 1 bloco será executado.

# Exemplo 3: Pode usar o brinquedo?

```
#include <stdio.h>

int main () {
    int altura_centimetros, idade;

    printf("Entre com a sua idade e altura (em centimetros): \n");
    scanf("%d %d", &idade, &altura_centimetros);

    if (idade < 12) {
        printf("Voce nao tem idade para usar o brinquedo!\n");
    }
    else {
        if (altura_centimetros < 150) {
            printf("Voce nao tem altura para usar o brinquedo!\n");
        }
        else {
            printf("Voce pode usar o brinquedo!\n");
        }
    }
    return 0;
}
```

# Comando Switch

- Permite selecionar uma entre várias ações alternativas
- Embora **if-else** possam executar testes para escolhas de uma entre várias alternativas, muitas vezes são deselegantes
- **Switch** tem um formato limpo e claro

# Comando Switch: Sintaxe

```
switch (variável) {
```

```
    case constante1:
```

```
        instrução1;
```

```
        instrução2;
```

```
        instrução3;
```

```
    break;
```

```
    case constante2:
```

```
        instrução1;
```

```
        instrução2;
```

```
        instrução3;
```

```
    break;
```

```
    ...
```

```
}
```

# Exemplo 4: Comando Switch

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

```
int main () {  
    int op;
```

```
    printf("Entre com uma opcao: \n");  
    scanf("%d", &op);
```

```
    switch(op) {  
        case 1:  
            printf("Opcao 1 selecionada!\n");  
            break;  
  
        case 2:  
            printf("Opcao 2 selecionada!\n");  
            break;  
  
        default:  
            printf("Opcao invalida! Entre com um valor valido!\n");  
            break;  
    }
```

```
    return 0;
```

```
}
```

```
Entre com uma opcao:
```

```
1
```

```
Opcao 1 selecionada!
```

```
Entre com uma opcao:
```

```
99
```

```
Opcao invalida! Entre com um valor valido!
```



# Operador Ternário

- Programadores usam o operador ternário em C para tomar decisões *inplace* de expressões condicionais **if-else**

```
#include <stdio.h>

int main () {
    int a = 10, b = 20, c;

    c = (a < b) ? a : b;

    printf("%d", c);

    return 0;
}
```

```
% ./a.out
10
```

# Estruturas de Repetição

# Laço For

- O laço **for** é geralmente usado quando queremos repetir algo por um número fixo de vezes
- Sabemos de antemão o número de vezes a repetir

```
#include <stdio.h>

int main () {
    int i;

    for (i = 0; i < 20; i++)
        printf("*");

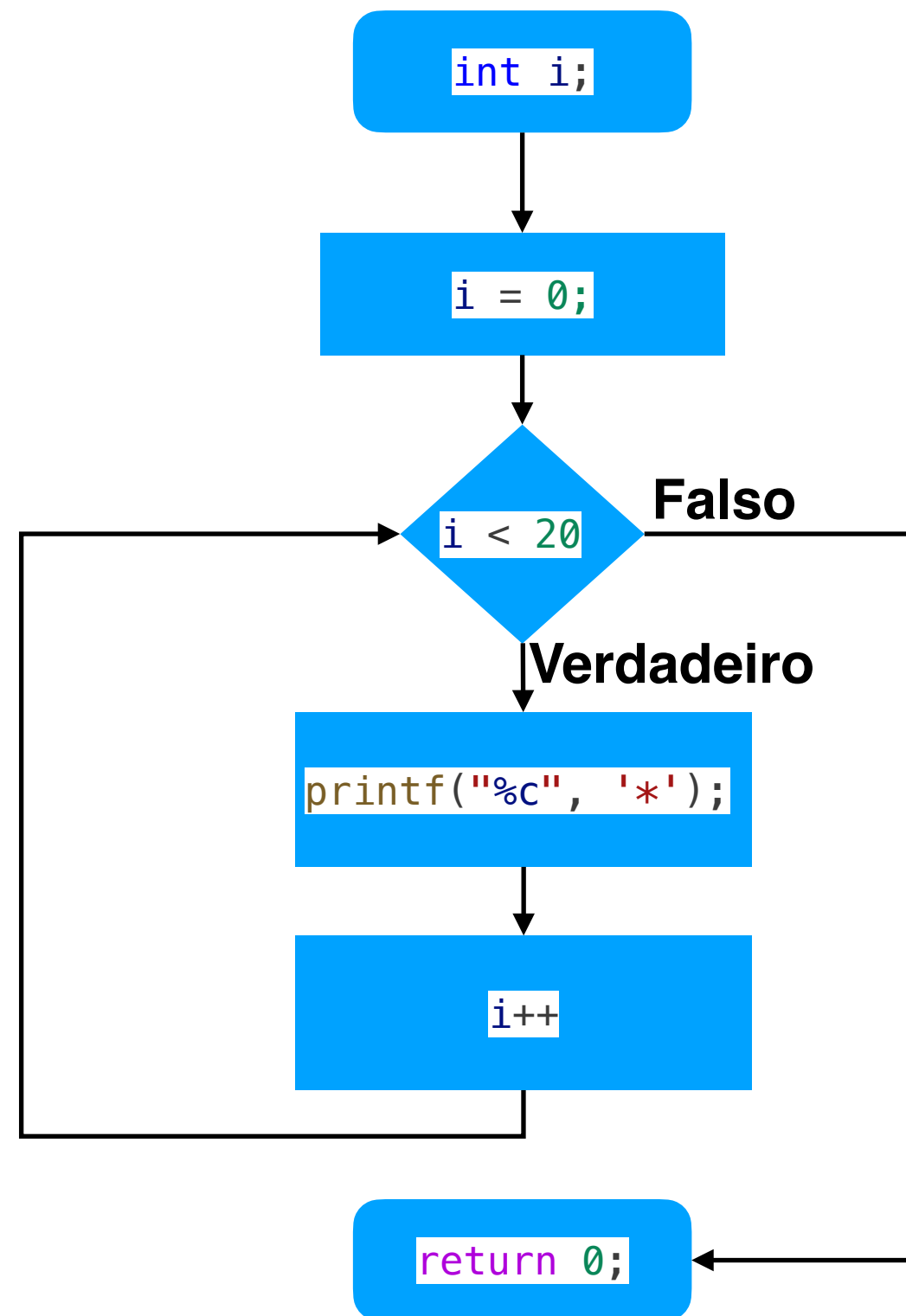
    printf("\n");

    return 0;
}
```

```
% ./a.out
*****
```

# Laço For (cont.)

```
1. #include <stdio.h>
2.
3. int main () {
4.     int i;
5.
6.     for(i = 0; i < 20; i++)
7.         printf("%c", '*');
8.
9.     printf("\n");
10.
11.     return 0;
12. }
```



# Sintaxe Inicialização: Exemplos


`for(i = 0;;) {`  **Inicialização da variável i com o valor 0.**

**Inicialização da variável i com o valor 0 e da variável j com 1.**

`for(i = 0, j = 1;;) {` 

`for(i = j;;) {`  **Inicialização da variável i com o valor de j.**

**Nenhuma inicialização é feita.**

`for(;;) {` 

# Sintaxe Condição: Exemplos

```
for( ; i < 20; ) {  
}
```

**Condição verifica se a variável i é menor que 20.**

**Expressão lógica verifica se a variável i é menor que 20 e também a variável j é maior que 10.**

```
for( ; i < 20 && j > 10; ) {  
}
```

```
for( ; i < j; ) {  
}
```

**Expressão lógica verifica se a variável i é menor que j.**

**Nenhuma expressão lógica. O laço será executado indefinidamente.**

```
for( ; ; ) {  
}
```

# Sintaxe Incremento: Exemplos

```
for(;; i += 1) {  
}
```

**Incremento de 1 à variável i  
ao final de cada execução  
do laço.**

**Incremento de 1 à variável i  
e -2 à variável j ao final de  
cada execução do laço.**

```
for(;; i += 1, j -= 2) {  
}
```

```
for(;; i++) {  
}
```

**Incremento de 1 à variável i  
ao final de cada execução  
do laço.**

**Nenhum incremento  
realizado ao final de cada  
execução do laço.**

```
for(;;) {  
}
```

# Exemplo 5: Imprime os pares

Apresente todos os números pares no intervalo de 600 a 1.

```
#include <stdio.h>

int main () {
    int i;

    for(i = 600; i > 1; i = i - 2)
        printf("%d", i);

    printf("\n");

    return 0;
}
```



# Operadores de Incremento e Decremento: Prefixado

```
n = 5;  
x = ++n;  
printf("n=%d; x=%d", n, x);
```



```
n = 5;  
n = n + 1;  
x = n;  
printf("n=%d; x=%d", n, x);
```

# Operadores de Incremento e Decremento: Pós-fixado

```
n = 5;  
x = n++;  
printf("n=%d; x=%d", n, x);
```



```
n = 5;  
x = n;  
n = n + 1;  
printf("n=%d; x=%d", n, x);
```

# Laço While

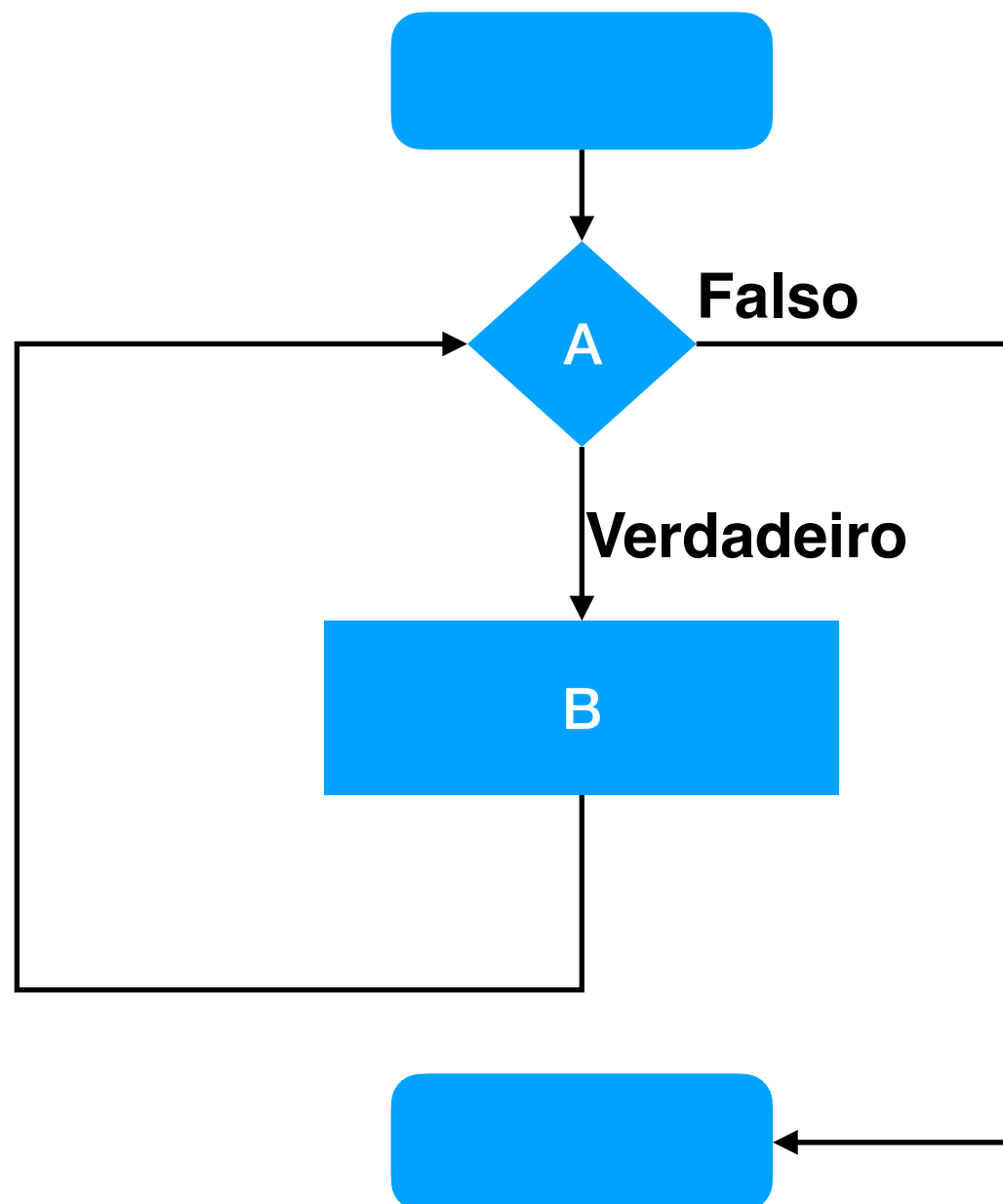
Analizando o comando:

- o comando se inicia com a palavra-chave **while**;
- a expressão-lógica deve estar entre parênteses;
- para a repetição de um conjunto de comandos é necessário colocá-lo dentro de um bloco de comandos, isto é, entre chaves: { };

```
while (<expressão-lógica>) {  
    comando1;  
    comando2;  
    ...  
}
```

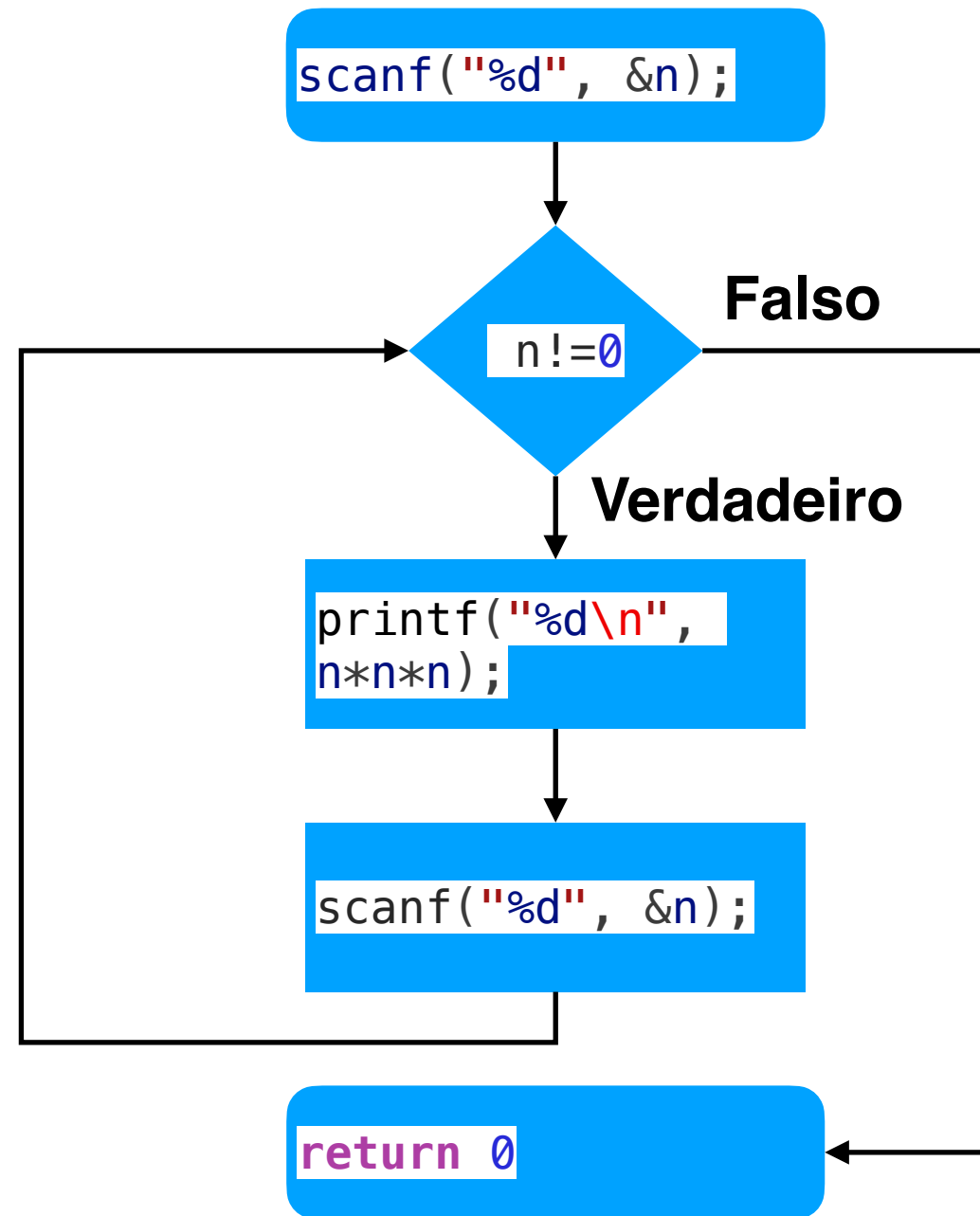
# Visualização Laço While

```
while(A)  
  B;
```



# Visualização Laço While (cont.)

```
1. #include <stdio.h>
2.
3. int main () {
4.     int n;
5.
6.     scanf("%d", &n);
7.
8.     while (n != 0) {
9.         printf("%d\n", n*n*n);
10.        scanf("%d", &n);
11.    }
12.
13.    return 0;
14.}
```



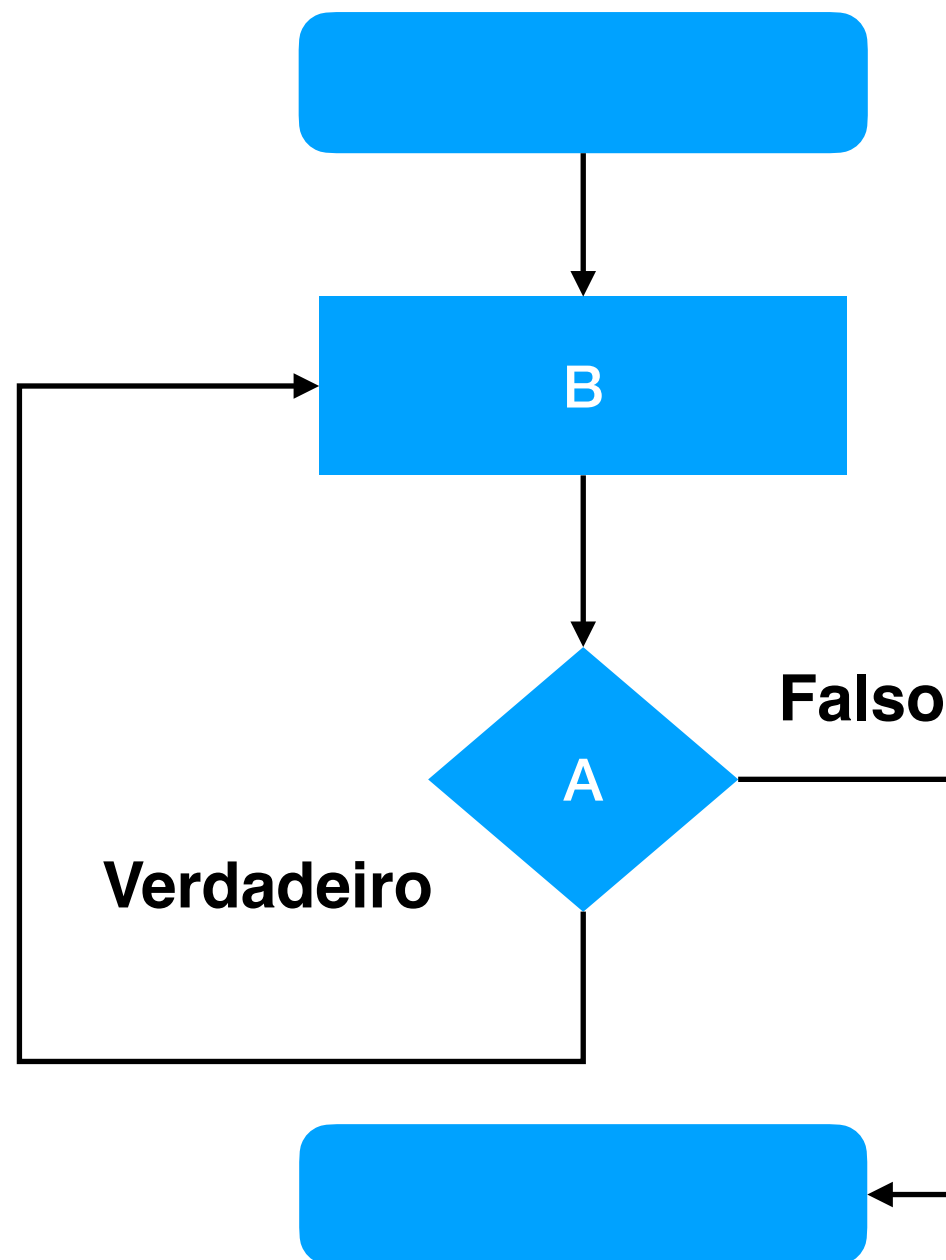
# Laço Do-While

- A diferença é que a condição para a repetição dos comandos é testada no final, somente depois de executá-los, pelo menos uma vez

```
<inicialização>;  
do {  
    ...  
    <incremento>;  
} while (<expressão-lógica>);
```

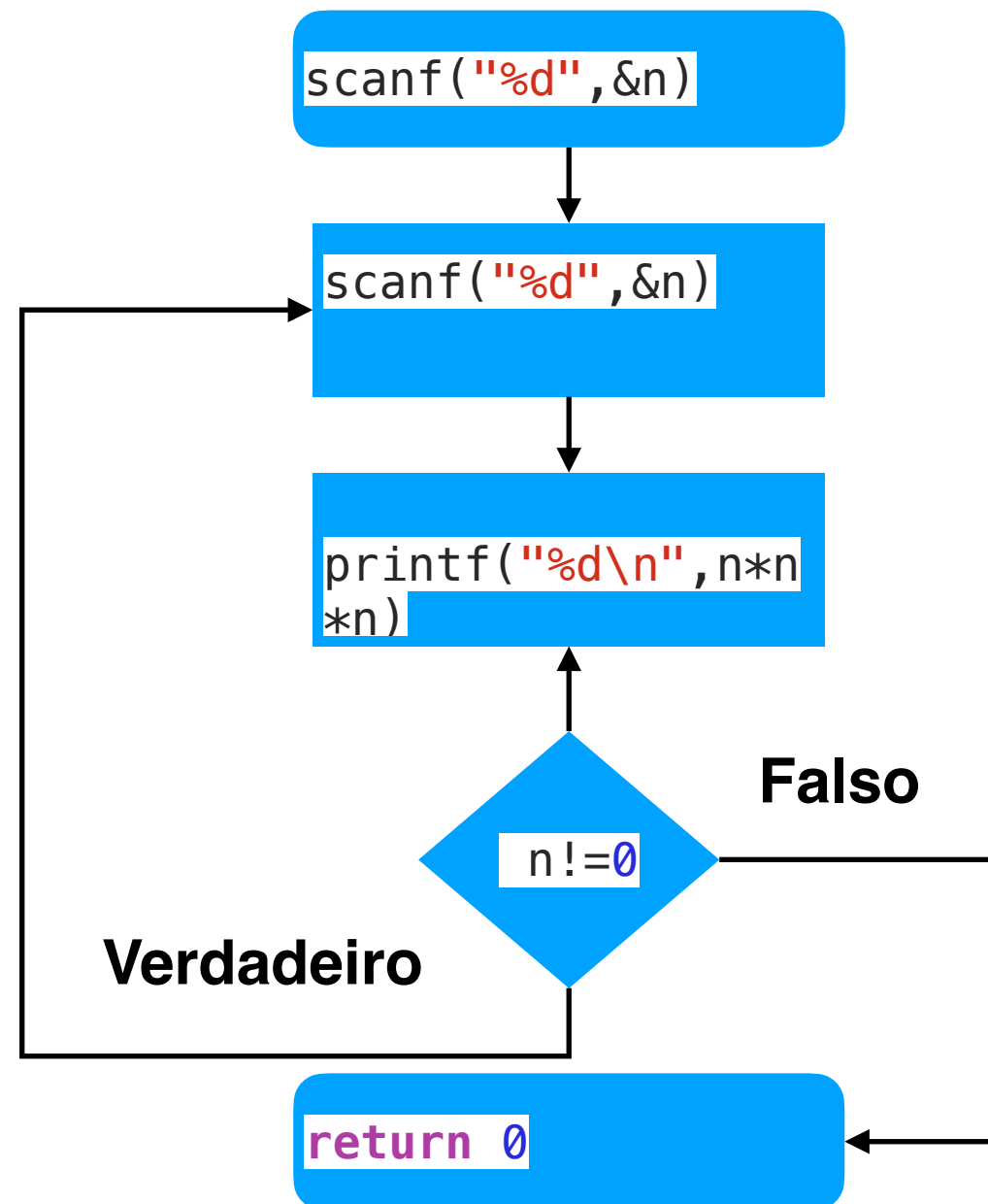
# Visualização Laço Do-While

```
do {  
    B;  
} while (A);
```



# Exemplo 6: Solução Do-While

```
1. #include <stdio.h>
2.
3. int main () {
4.     int n;
5.
6.     do {
7.         scanf("%d", &n);
8.         printf("%d", n*n*n);
9.     } while(n != 0);
10.
11.     return 0;
12. }
13.
```





# Referências

- DEITEL, Paul; DEITEL, Harvey; C++ How to Program (9 edition). Pearson, 2016.