

Linguagens de Programação 1

```
int aula = 3;  
printf("\aAula #%d - Instrucoes de Controlo\n", aula * 2 - 2);
```

Message of the Day

"Talk is cheap. Show me the code."

"Software is like sex: it's better when it's free."

"A computer is like air conditioning - it becomes useless when you open Windows."

"Excusing bad programming is a shooting offence, no matter what the circumstances."

— Linus Torvalds




Linus Torvalds (1969–)

Criador do Linux, o sistema operativo open source mais usado no mundo



Criou o Git, sistema de versionamento que revolucionou o desenvolvimento de software

Conteúdo

Instruções de Seleção

- i. `if`, `else`, `else if` 
- ii. `switch` 
- iii. Operador ternário 

Estruturas de Repetição

- i. `while`, `do while`, `for` 
- ii. `break` e `continue` 

Funções e scope de variáveis

Instruções de Controlo de Fluxo

Alteram a execução sequencial de um programa

Instruções condicionais (`if` , `else` , `switch`)

Ciclos (`while` , `for` , `do-while`)

Instrução **if**

Permite executar um bloco de código apenas se uma condição for verdadeira. 

```
int x = 10;

if (x <= 10) {
    printf("X é menor ou igual a 10\n");
}
```

```
desconto = 0;

if (idade >= 16 && idade <= 25) // desconto jovem
    desconto += 10;

desconto += 5; // desconto para todos

printf("O seu desconto é: %d\n", desconto);
```

Uso opcional de `{}`


Em C, se um bloco de código contém apenas uma instrução, as `{}` podem ser omitidas!

```
int x = 10;  
if (x > 5)  
    printf("X é maior que 5\n"); // Sem chaves!
```

 Isso vale para:

`if`, `else`, `else if`

`while`, `for`, `do while`

 **Melhor prática:** Sempre usar `{}` para evitar ambiguidades!

Mas cuidado!

Se adicionar uma segunda instrução, o comportamento pode mudar inesperadamente:

```
if (x > 5)  
    printf("X é maior que 5\n");  
    printf("Esta linha será sempre executada!\n"); // Fora do if!
```

Instrução **if-else**

A cláusula **else** é opcional e executada se a condição for falsa. ✗

```
int nota = 5;
if (nota >= 10) {
    printf("Aprovado\n");
} else {
    printf("Reprovado\n"); // executa este bloco se a condição for falsa
}
```

Instrução **if-else if**

Permite testar múltiplas condições. 

```
int idade = 18;

if (idade < 5) {
    printf("Isento\n");
} else if (idade < 18) {
    printf("Tarifa criança\n");
} else if (idade < 65) {
    printf("Tarifa normal\n");
} else {
    printf("Tarifa sénior\n");
}
```

? Qual será a saída se **idade = 65** ?

- (A) "Tarifa criança"
- (B) "Tarifa normal"
- (C) "Tarifa sénior"
- (D) Nenhuma saída

Instruções **if** Aninhadas

Como funciona um **if** dentro de outro **if** ?

```
if (condição1) {  
    if (condição2) {  
        // código executado se ambas forem verdadeiras  
    }  
}
```

✓ Pode ser substituído por um **&&** :

```
if (condição1 && condição2) {  
    // código executado se ambas forem verdadeiras  
}
```

 Usar **&&** pode tornar o código mais limpo e direto!

? Operador Ternário

Uma alternativa curta ao `if-else` . ✨

```
exp1 ? exp2 : exp3
```

- i. `exp1` é avaliada
- ii. se for verdadeira, devolve `exp2`
- iii. se for falsa, devolve `exp3`

```
int a = 5, b = 10;  
int maior = (a > b) ? a : b;  
printf("Maior: %d\n", maior);
```

✓ Simples e eficaz para expressões curtas.

? Operador Ternário (exemplos)

```
int x = 5, y;  
y = x > 6 ? 7 : 8;
```

```
int x = 9, y;  
y = x > 6 ? 7 : 8
```

```
x = (a == b) ? 0 : (a > b) ? a : b;
```

Qual o valor de **x** quando:

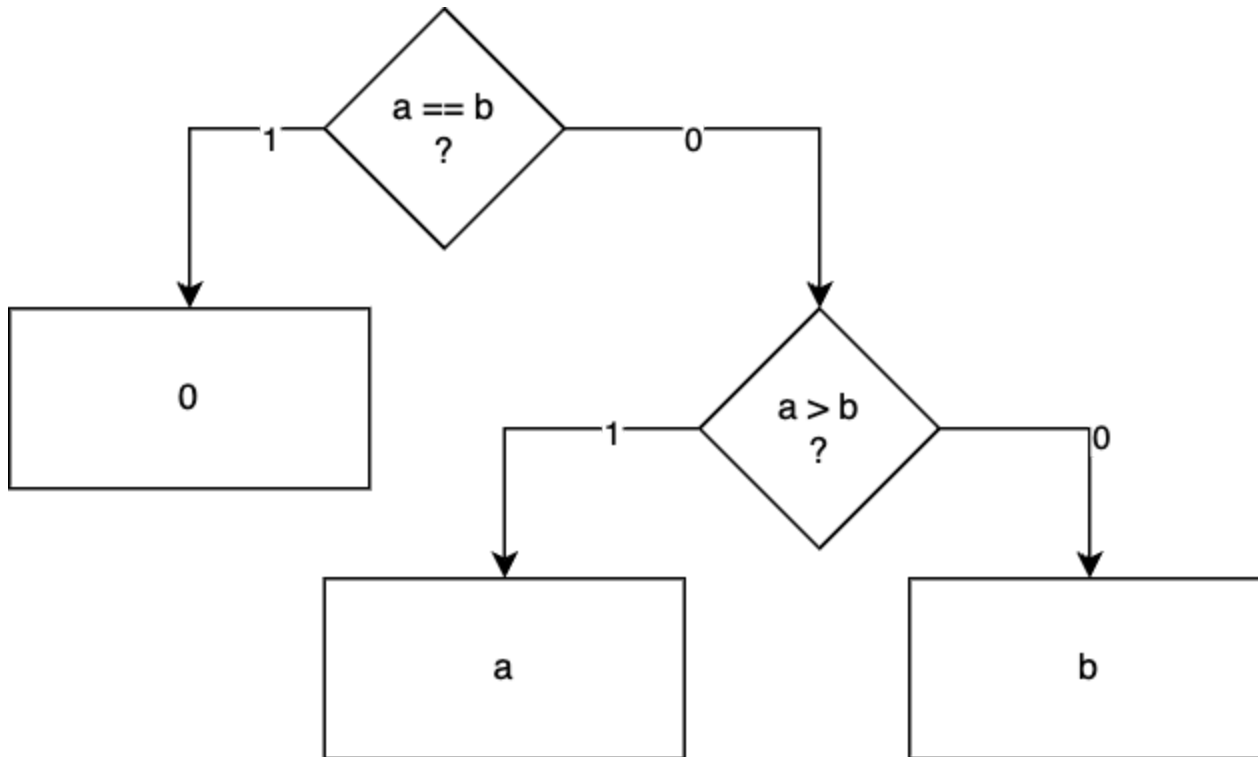
- i. **a = 5** e **b = 10**
- ii. **a = 7** e **b = 7**
- iii. **a = -4** e **b = -10**

R: **10** , **0** , **-4**

? Operador Ternário - fluxograma

💡 Quando a operação é complexa devemos fazer um **fluxograma**

```
x = (a == b) ? 0 : (a > b) ? a : b;
```



Instrução `switch`

*Avalia uma expressão e compara com os `case` disponíveis. ⚖️

```
int x = 2;
switch (x) {
    case 0:
        printf("Zero\n");
        break;
    case 1:
        printf("Um\n");
        break;
    case 2:
        printf("Dois\n");
        break;
    default: // default funciona como o else
        printf("Outro valor\n");
}
```

✅ Apenas valores inteiros (`int` , `char`)
podem ser testados.

⚠️ Sem `break` , a execução continua para
o próximo `case` !

Instrução **switch** (cont)

Escreva uma função que recebe um character e retorna 0 se for consoante, 1 se for vogal

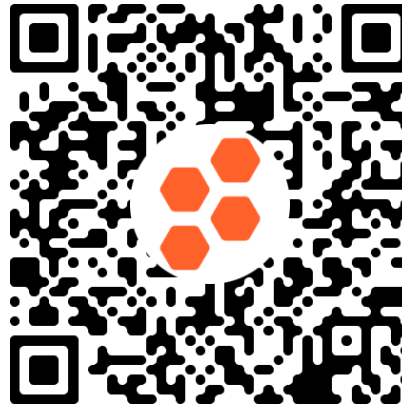
```
#include <ctype.h>
// para podermos usar a funcao tolower()

int isVowel(char c) {
    switch(tolower(c)) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
            return 1;
    }
    return 0;
}
```

```
int isVowel(char c) {
    char lower_c = tolower(c);
    if (lower_c == 'a' || lower_c == 'e' ||
        lower_c == 'i' || lower_c == 'o' ||
        lower_c == 'u')
        return 1;

    return 0;
}
```

? Quizz Instruções de Seleccão



No campo nome devem colocar o número de aluno 2XXXXXXX.

Estruturas de Repetição: **while**

Executa enquanto a condição for verdadeira.

```
int num = 0;
while (num < 5) {
    printf("%d\n", num);
    num++;
}
```

✓ Útil quando o número de iterações não é conhecido previamente.

Estruturas de Repetição: **do-while**

Garante pelo menos uma execução antes da condição ser verificada.

```
int num, r = 0;

do {
    puts("insira um numero inteiro positivo");
    r = scanf ("%d", &num);
} while (r != 1 || num < 0);

printf("Valor valido lido: %d\n", num);
```

✓ Útil para validação de inputs do utilizador.

neste exemplo, o ciclo só repete se o utilizador não introduzir um número inteiro, ou se introduzir um número negativo.

```
insira um numero inteiro positivo
-10
insira um numero inteiro positivo
5
Valor valido lido: 5
```

Estruturas de Repetição: **for**

Ideal para loops com contagem definida.

```
for (int i = 0; i < 5; i++) {  
    printf("%d\n", i);  
}
```

✓ Possui inicialização, condição e atualização no cabeçalho.

A inicialização e atualização podem ter várias instruções, desde que separadas por `,`.

```
int notas[10], i, j;  
double media;  
  
for (i = 0, media = 0, j = 1 ; i < 10 ; i++, j++)  
{  
    media += notas[i];  
}  
  
media /= i;
```

Inicialização, condição e actualização do `for` são opcionais:

```
for (; i < 10 ;)  
    media += notas[i++];
```

```
for (;;) // ciclo infinito  
    printf("EU VOU APRENDER A USAR O FOR\n");
```

Instruções de Controlo de Ciclo

✓ Úteis para controlo avançado de loops.

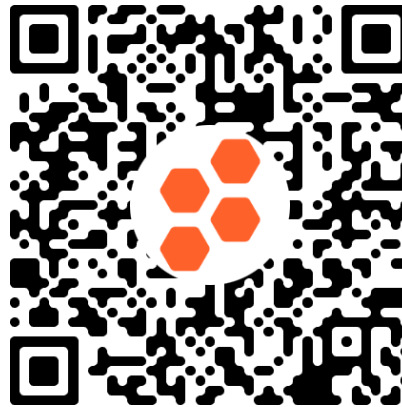
break : Sai imediatamente do loop/switch. 

```
for (int i = 1; i <= 5; i++) {  
    if (i == 3) break; // imprime os números até encontrar o 3  
    printf("%d\n", i);  
}
```

continue : Salta para a próxima iteração. 

```
for (int i = 1; i <= 5; i++) {  
    if (i == 3) continue; // imprime os números até 5, mas não imprime o 3  
    printf("%d\n", i);  
}
```

? Quizz Estruturas de Repetição




No campo nome devem colocar o número de aluno 2XXXXXXX.

Funções e Scope de Variáveis

Invocação de Funções

Chamamos a função pelo seu nome, passando argumentos.

```
int maior = max(10, 20);  
printf("Maior: %d\n", maior);
```

 O programa **pausa** a execução da função chamadora e só continua após o `return` da função invocada!

Passagem de Parâmetros

✓ Parâmetros podem ser **passados por valor**, ou seja, cópias dos valores originais.

```
void altera(int x) {  
    x = 50;  
}  
  
int main() {  
    int num = 10;  
    altera(num);  
    printf("Num: %d\n", num); // Saída: 10  
    return 0;  
}
```

 **Atenção:** O valor de `num` não é alterado na função `altera` !

Escopo de Variáveis

Variáveis Globais

Definidas **fora** de qualquer função, acessíveis por todo o programa.

```
int global = 10;
void funcao() {
    printf("Global: %d\n", global);
}
```

✅ São úteis, mas devem ser **evitadas** para manter o código modular!

❌ Nesta UC não serão permitidas variáveis globais

Variáveis Locais

Definidas **dentro** de uma função, existem apenas no seu scope. A variável existe e é visível **apenas** dentro da **função** depois de ser declarada e só **enquanto** a função estiver a **ser** executada.

```
void funcao() {  
    int local = 20;  
    printf("Local: %d\n", local);  
}
```

 Fora da função, a variável `local` não existe!

Variáveis Locais (cont.)

os parâmetros de uma função são também variáveis locais

```
int mediana(int a, int b, int c)
{
    // a, b, c são variáveis locais
    return (a > b) ? ((b > c) ? b : (a > c) ? c : a)
           : ((a > c) ? a : (b > c) ? c : b);
}
```

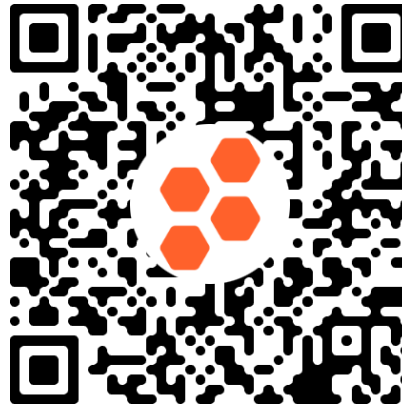
Variáveis Estáticas

✓ Variáveis **static** mantêm o valor entre chamadas da função.

```
void contador() {  
    static int count = 0;  
    count++;  
    printf("Contagem: %d\n", count);  
}  
  
int main() {  
    contador(); // Contagem: 1  
    contador(); // Contagem: 2  
    return 0;  
}
```

 **static** é útil para preservar estado dentro de funções!

? Quizz - Funções e Variáveis



No campo nome devem colocar o número de aluno 2XXXXXXX.

? Q&A

 Dúvidas?