



CONTROLE DE FLUXO

ECM404

ESTRUTURA CONDICIONAL – IF

- ❑ Na linguagem C, o comando **if** é utilizado quando for necessário executar um comando sujeito ao resultado de um teste. A estrutura de um comando **if** é:

```
if( condição ){  
    sequência de comandos;  
}
```

- ❑ A **condição** poderá assumir dois valores:
 - ❑ Falso ou 0 – Neste caso, a **sequência de comandos** não será executada;
 - ❑ Verdadeiro ou valores diferentes de zero – Neste caso, a **sequência de comandos** será executada.

OPERADORES RELACIONAIS E LÓGICOS

❑ Operadores Relacionais:

Operador	Descrição
==	Igualdade
!=	Diferença
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual

❑ Operadores Lógicos:

Operador	Descrição
&&	Lógica E (AND)
	Lógica OU (OR)
!	Inversão (NOT)

OPERADORES RELACIONAIS E LÓGICOS

```
int main (){
    int a = 10;
    int b = 2;
    printf("2 == 10 = %i\n",b==a);
    printf("2 != 10 = %i\n",b!=a);
    printf("2 > 10 = %i\n",b>a);
    printf("2 < 10 = %i\n",b<a);
    printf("2 >= 10 = %i\n",b>=a);
    printf("2 <= 10 = %i\n",b<=a);

    a = 1;
    b = 0;
    printf("a && b = %i\n",a&&b);
    printf("a || b = %i\n",a||b);
    printf("!a = %i\n",!a);
    printf("!b = %i\n",!b);

    return 0;
}
```

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
PS D:\lista1> make ex1 run			
mkdir -p bin			
gcc src/ex1.c -Llib -lm -o bin/ex1.out			
2 == 10 = 0			
2 != 10 = 1			
2 > 10 = 0			
2 < 10 = 1			
2 >= 10 = 0			
2 <= 10 = 1			
a && b = 0			
a b = 1			
!a = 0			
!b = 1			
PS D:\lista1> █			

EXEMPLO IF

```
int main(){  
    int num;  
    printf("Digite um numero: ");  
    scanf("%i", &num);  
    if( num > 10 ){  
        printf("Numero maior que 10!\n");  
    }  
    return 0;  
}
```

ESTRUTURA CONDICIONAL – IF-ELSE

- ❑ A forma geral de um comando **if-else** é:

```
if( condição ){  
    sequência de comandos 1;  
}  
else{  
    sequência de comandos 2;  
}
```

- ❑ A **condição** poderá assumir dois valores:
 - ❑ Falso – A **sequência de comandos 2** será executada;
 - ❑ Verdadeiro – A **sequência de comandos 1** será executada.

EXEMPLO IF-ELSE

```
int main(){
    int num;
    printf("Digite um numero: ");
    scanf("%i", &num);
    if( num > 10 ){
        printf("Numero maior que 10!\n");
    }
    else{
        printf("Numero menor ou igual a 10!\n");
    }
    return 0;
}
```

ESTRUTURA CONDICIONAL – IF-ELSE

- ❑ Tanto no comando **if** quanto no **else** as chaves podem ser ignoradas se a instrução contida for única:

```
if( num == 10 ) // if sem chaves
    printf("Numero igual a 10");
else           // else sem chaves
    printf("Numero diferente de 10");
```

```
if( num == 10 ){ // if com chaves
    printf("Numero igual a 10");
}
else           // else sem chaves
    printf("Numero diferente de 10");
```

```
if( num == 10 ) // if sem chaves
    printf("Numero igual a 10");
else{           // else com chaves
    printf("Numero diferente de 10");
}
```


ESTRUTURA CONDICIONAL – ANINHAMENTO

```
if( condição ){  
    if(condição){  
        instrução 1;  
        ...  
        instrução N;  
    }  
    else{  
        instrução 1;  
        ...  
        instrução N;  
    }  
}  
else{  
    instrução 1;  
    ...  
    instrução N;  
}
```

```
if( condição ){  
    instrução 1;  
    ...  
    instrução N;  
}  
else{  
    if(condição){  
        instrução 1;  
        ...  
        instrução N;  
    }  
    else{  
        instrução 1;  
        ...  
        instrução N;  
    }  
}
```

EXEMPLO ANINHAMENTO

```
int main(){
    char ch;
    printf("Digite um numero de 0 a 9: ");
    scanf("%c", &ch);

    if( ch == '0' ) printf("Zero\n");
    else if( ch == '1' ) printf("Um\n");
    else if( ch == '2' ) printf("Dois\n");
    ...
    else if( ch == '9' ) printf("Nove\n");
    else
        printf("Não era um numero\n");

    return 0;
}
```

ESTRUTURA CONDICIONAL – SWITCH

- ❑ O comando **switch** é próprio para se testar uma variável em relação a diversos valores pré-estabelecidos;
- ❑ Semelhante ao **if-else-if**, porém não aceita expressões, apenas **constantes**;
- ❑ O switch testa a variável e executa a os comandos, cujo “**case**” corresponda ao valor atual da variável, até encontrar um “**break**”.

ESTRUTURA CONDICIONAL – SWITCH

- ❑ Estrutura do comando **switch** é:

```
switch (expressão){  
    case [valor 1]:  
        sequencia de comandos 1;  
        break;  
    case [valor 2]:  
        sequencia de comandos 2;  
        break;  
    ...  
    case [valor n]:  
        sequencia de comandos n;  
        break;  
    default:  
        sequencia de comandos caso nenhuma outra seja satisfeita;  
        break;  
}
```

EXEMPLO SWITCH

```
int main (){
    char ch;
    printf("Digite um numero de 0 a 9: ");
    scanf("%c",&ch);

    switch (ch){
        case '0': printf("Zero\n"); break;
        case '1': printf("Um\n"); break;
        case '2': printf("Dois\n"); break;
        ...
        case '9': printf("Nove\n"); break;
        default: printf("Nao era um numero\n"); break;
    }

    return 0;
}
```

ESTRUTURAS DE REPETIÇÃO

- ❑ Uma estrutura de repetição permite que uma sequência de comandos seja executada repetidamente, enquanto determinadas condições são satisfeitas;
- ❑ Essas condições são representadas por expressões lógicas (como, por exemplo, $A > B$, $C < 10$, etc)
- ❑ As estruturas de repetição podem ser:
 - ❑ Repetição com Teste no Início;
 - ❑ Repetição com Teste no Fim;
 - ❑ Repetição Contada.

REPETIÇÃO COM TESTE NO INÍCIO

- ❑ O comando **while** é utilizado para repetir um conjunto de comandos **enquanto** a condição for **verdadeira**; A estrutura de um comando **while** é:

```
while( condição ){  
    sequência de comandos;  
}
```

```
int main (){  
    int a = 0;  
    while(a<100){  
        printf("%i\n",a);  
        a++;  
    }  
    return 0;  
}
```

REPETIÇÃO COM TESTE NO FIM

- ❑ O comando **do-while** também é utilizado para repetir um conjunto de comandos **enquanto** a condição for **verdadeira**. A estrutura de um comando **do-while** é:

```
do{  
    sequência de comandos;  
}  
while( condição );
```

- ❑ Note que, diferentemente do comando **while**, a sequência de comandos do **do-while** é executada ao menos uma vez.

```
int main (){  
    int a = 0;  
    do{  
        printf("%i\n",a);  
        a++;  
    }  
    while(a<100);  
    return 0;  
}
```


REPETIÇÃO CONTADA

- ❑ O comando **for** também é utilizado para repetir um conjunto de comandos **N vezes** até que a condição deixe de ser **verdadeira**. A estrutura de um comando **for** é:

```
for(inicialização; condição; incremento){  
    sequência de comandos;  
}
```

REPETIÇÃO CONTADA

```
for(inicialização; condição; incremento){  
    sequência de comandos;  
}
```

- ❑ **Inicialização:** iniciar variáveis (contador);
- ❑ **Condição:** avalia a condição. Se verdadeiro, executa a sequência de comandos, senão encerra o laço;
- ❑ **Incremento:** ao término da sequência de comandos, incrementa o valor do contador;

```
int main (){  
    int a;  
    for(a=0; a<100; a++){  
        printf("%i\n",a);  
    }  
    return 0;  
}
```

ESTRUTURAS DE REPETIÇÃO - RESUMO

- ❑ Comando **while**: repete uma sequência de comandos **enquanto** uma condição for verdadeira;
- ❑ Comando **for**: repete uma sequência de comandos **N vezes**;

COMANDO BREAK

- ❑ O comando **break** serve para:
 - ❑ Quebrar a execução de um comando (como no caso do switch);
 - ❑ Interromper a execução de qualquer repetição (while, do-while ou for).

```
int main (){  
    int a = 0;  
    while(a<100){  
        if(a==25)  
            break;  
        printf("%i\n",a);  
        a++;  
    }  
    return 0;  
}
```

COMANDO CONTINUE

- ❑ O comando **continue**:
 - ❑ Diferente do **break**, só funciona dentro da repetição;
 - ❑ Pula a iteração atual da repetição.

```
int main (){  
    int a = 0;  
    while(a<100){  
        if(a==25)  
            continue;  
        printf("%i\n",a);  
        a++;  
    }  
    return 0;  
}
```

Cuidado! Todos os comandos que estiverem após o **continue** serão ignorados. Assim, **a** não será incrementado e o programa ficará em um laço infinito.

COMANDO CONTINUE

- ❑ O comando **continue**:
 - ❑ Diferente do **break**, só funciona dentro da repetição;
 - ❑ Pula a iteração atual da repetição.

```
int main (){  
    int a = 0;  
    while(a<100){  
        a++;  
        if(a==25)  
            continue;  
        printf("%i\n",a);  
    }  
    return 0;  
}
```

LISTA DE EXERCÍCIOS

- ❑ Abra a pasta Downloads no VS Code
 - ❑ Utilize o comando `git clone https://github.com/ECM404/lista2.git --recursive`
 - ❑ Entre na pasta utilizando o comando `cd lista2`
 - ❑ Inicialize o diretório com o comando `make install`

PARA RELEMBRAR!

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Avell 1711 IRON\Downloads> git clone https://github.com/ECM404/lista1.git --recursive
```

```
Cloning into 'lista1'...
remote: Enumerating objects: 22, done.
remote: Counting objects: 100% (22/22), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 22 (delta 4), reused 21 (delta 3), pack-reused 0
Receiving objects: 100% (22/22), done.
Resolving deltas: 100% (4/4), done.
Submodule 'tests/check404' (https://github.com/ECM404/check404.git) registered for path 'tests/check404'
Cloning into 'C:/Users/Avell 1711 IRON/Downloads/lista1/tests/check404'...
remote: Enumerating objects: 39, done.
remote: Counting objects: 100% (39/39), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 39 (delta 13), reused 30 (delta 9), pack-reused 0
Receiving objects: 100% (39/39), 6.46 KiB | 1.62 MiB/s, done.
Resolving deltas: 100% (13/13), done.
Submodule path 'tests/check404': checked out '08a58596a61198cabd9f6caebd437a3bd29fd0a4'
```

```
PS C:\Users\Avell 1711 IRON\Downloads> cd lista1
```

```
PS C:\Users\Avell 1711 IRON\Downloads\lista1> make install
```

```
make[1]: Entrando no diretório '/cygdrive/c/Users/Avell 1711 IRON/Downloads/lista1/tests/check404'
```

```
python -m pip install -r requirements.txt
```

```
WARNING: Ignoring invalid distribution -ip (c:\python39\lib\site-packages)
```

```
WARNING: Ignoring invalid distribution -ip (c:\python39\lib\site-packages)
```

```
Requirement already satisfied: pyyaml in c:\python39\lib\site-packages (from -r requirements.txt (line 1)) (6.0)
```

```
Requirement already satisfied: pexpect in c:\python39\lib\site-packages (from -r requirements.txt (line 2)) (4.8.0)
```

```
Requirement already satisfied: wxexpect in c:\python39\lib\site-packages (from -r requirements.txt (line 3)) (4.0.0)
```

```
Requirement already satisfied: coloredlogs in c:\python39\lib\site-packages (from -r requirements.txt (line 4)) (15.0.1)
```

```
Requirement already satisfied: verboselogs in c:\python39\lib\site-packages (from -r requirements.txt (line 5)) (1.7)
```

```
Requirement already satisfied: ptyprocess>=0.5 in c:\python39\lib\site-packages (from pexpect->-r requirements.txt (line 2)) (0.7.0)
```

```
Requirement already satisfied: pywin32>=220 in c:\python39\lib\site-packages (from wxexpect->-r requirements.txt (line 3)) (303)
```

```
Requirement already satisfied: psutil>=5.0.0 in c:\python39\lib\site-packages (from wxexpect->-r requirements.txt (line 3)) (5.9.0)
```

```
Requirement already satisfied: humanfriendly>=9.1 in c:\python39\lib\site-packages (from coloredlogs->-r requirements.txt (line 4)) (10.0)
```

```
Requirement already satisfied: pyreadline3 in c:\python39\lib\site-packages (from humanfriendly>=9.1->coloredlogs->-r requirements.txt (line 4)) (3.4.1)
```

```
WARNING: Ignoring invalid distribution -ip (c:\python39\lib\site-packages)
```

```
WARNING: Ignoring invalid distribution -ip (c:\python39\lib\site-packages)
```

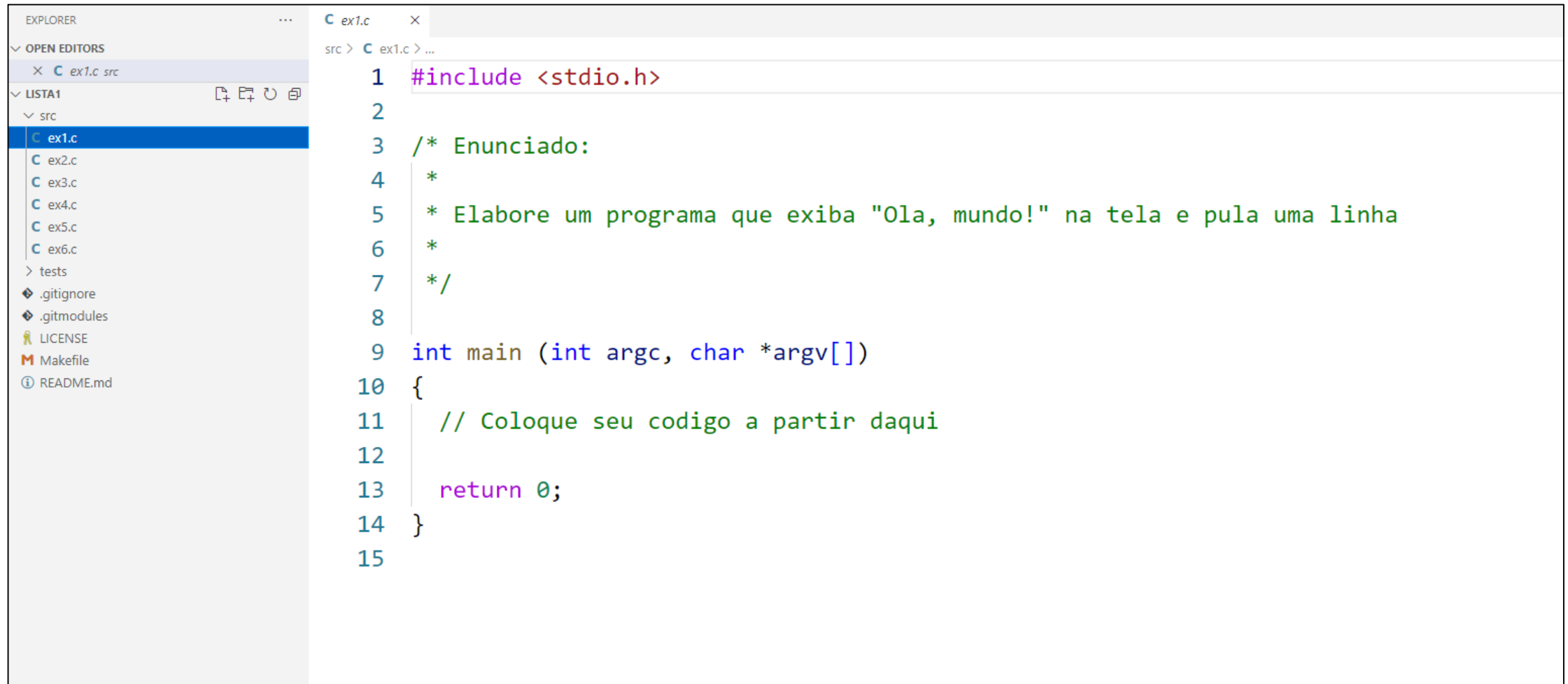
```
WARNING: Ignoring invalid distribution -ip (c:\python39\lib\site-packages)
```

```
make[1]: Saindo do diretório '/cygdrive/c/Users/Avell 1711 IRON/Downloads/lista1/tests/check404'
```

```
PS C:\Users\Avell 1711 IRON\Downloads\lista1> █
```


PARA RELEMBRAR!

- ❑ Abra a pasta Lista1 no VS Code e em src estarão os exercícios



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure with a folder named 'LISTA1' containing a subfolder 'src'. Inside 'src', several C files are listed: 'ex1.c' (selected), 'ex2.c', 'ex3.c', 'ex4.c', 'ex5.c', and 'ex6.c'. Other files like '.gitignore', '.gitmodules', 'LICENSE', 'Makefile', and 'README.md' are also visible. The main editor window shows the content of 'ex1.c', which is a C program template. The code includes a header for stdio.h, a multi-line comment in Portuguese describing the task (writing a program that prints 'Ola, mundo!' and a blank line), and a basic 'main' function structure with a return statement.

```
1 #include <stdio.h>
2
3 /* Enunciado:
4  *
5  * Elabore um programa que exiba "Ola, mundo!" na tela e pula uma linha
6  *
7  */
8
9 int main (int argc, char *argv[])
10 {
11     // Coloque seu codigo a partir daqui
12
13     return 0;
14 }
15
```

PARA RELEMBRAR!

- ❑ Para rodar apenas um exercício, certifique-se que está no diretório raiz da lista e execute o comando `make [nome do ex] run`.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
PS C:\Users\Avell 1711 IRON\Downloads\lista1> make ex1 run
```

- ❑ Para realizar um grupo de testes, certifique-se que está no diretório raiz da lista e execute o comando `make test`

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
PS C:\Users\Avell 1711 IRON\Downloads\lista1> make test
```