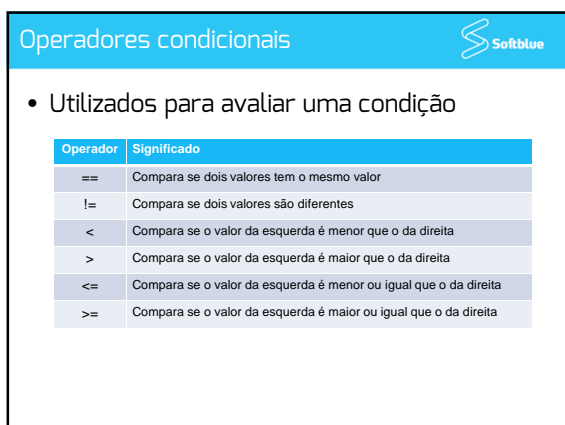




Linguagem C
Controle de fluxo e repetição

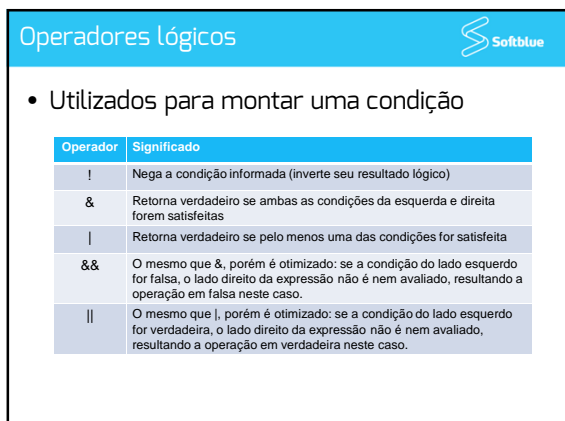
Softblue
cursos online



Operadores condicionais

- Utilizados para avaliar uma condição

Operador	Significado
==	Compara se dois valores tem o mesmo valor
!=	Compara se dois valores são diferentes
<	Compara se o valor da esquerda é menor que o da direita
>	Compara se o valor da esquerda é maior que o da direita
<=	Compara se o valor da esquerda é menor ou igual que o da direita
>=	Compara se o valor da esquerda é maior ou igual que o da direita



Operadores lógicos

- Utilizados para montar uma condição

Operador	Significado
!	Nega a condição informada (inverte seu resultado lógico)
&	Retorna verdadeiro se ambas as condições da esquerda e direita forem satisfeitas
	Retorna verdadeiro se pelo menos uma das condições for satisfeita
&&	O mesmo que &, porém é otimizado: se a condição do lado esquerdo for falsa, o lado direito da expressão não é nem avaliado, resultando a operação em falsa neste caso.
	O mesmo que , porém é otimizado: se a condição do lado esquerdo for verdadeira, o lado direito da expressão não é nem avaliado, resultando a operação em verdadeira neste caso.

Comando if



- `if(condição) { } [else if(condição) { }] [else { }]`
 - Permite executar um bloco de código se determinada condição for verdadeira

```
if(3 > 5) {
    // Não entra no primeiro if
}

if(1 < 10) {
    // Entra no segundo if
}
else {
    // Não entra no else do segundo if
}
```

```
int i = 5;
if(i == 3)
{
    // O valor de i é 3
}
else if(i == 4)
{
    // O valor de i é 4
}
else {
    // O valor de i não é 3 nem 4.
}
```

Comando short if



- `condição ? verdadeiro : falso;`
 - Forma reduzida de escrever o comando if
 - Deve ser utilizado somente em casos onde o bloco de código for curtíssimo

```
int x = 5;
int y;

if(x % 2 == 0) {
    // Caso x seja par
    y = 60;
}
else {
    // Caso x seja impar
    y = 61;
}
```

```
int x = 5;
int y;

y = x % 2 == 0 ? 60 : 61;

// Se x for par, atribuirá 60, caso contrário 61
```

Comando switch



- `switch(valor inteiro) { cases }`
 - Permite executar um bloco de código específico dependendo do valor avaliado

```
int i = 1;
switch(i)
{
    case 0:
        // O valor de i é 0
        break;
    case 1:
        // O valor de i é 1
        break;
    default:
        // Nenhum
        break;
}
```

Comando for



- `for(inicialização; condição; incremento) { }`
 - Permite criar um laço de repetição, executando para cada valor o mesmo bloco de código

```
for(int i = 0; i < 10; i++)  
{  
    // Exibir i  
}  
  
// Resultado: 0 1 2 3 4 5 6 7 8 9
```

Comando while



- `while(condição) { }`
 - Permite executar várias vezes um bloco de código enquanto sua condição for verdadeira

```
int i = 0;  
while(i < 5)  
{  
    // Exibir i  
    i++;  
}  
  
// Resultado: 0 1 2 3 4
```

Comando do while



- `do { } while (condição);`
 - Similar ao while, com a diferença de que o do while tem o bloco de código executado obrigatoriamente uma vez antes de avaliar a condição

```
int i = 0;  
do  
{  
    // Exibir i  
    i++;  
}  
while(i < 5);  
  
// Resultado: 0 1 2 3 4
```

Comando break



- **break**
 - Permite interromper o comando de repetição e avançar para o próximo comando

```
for(int i = 0; i < 10; i++)
{
    if(i == 4)
    {
        break;
    }
    // Exibir i
}
// Resultado: 0 1 2 3
```

Comando continue



- **continue**
 - Permite instruir ao comando de repetição que interrompa a iteração atual e avance para a próxima iteração da repetição

```
for(int i = 0; i < 10; i++)
{
    if(i == 4)
    {
        continue;
    }
    // Exibir i
}
// Resultado: 0 1 2 3 5 6 7 8 9
```

Comando return / exit



- **return**
 - Encerra a execução da função
- **exit**
 - Encerra a aplicação
 - Biblioteca [stdlib.h](#)

```
void funcaoQualquer()
{
    /* ... tarefas ... */
    return;
}

int main()
{
    funcaoQualquer();
}
```

```
void funcaoQualquer()
{
    /* ... tarefas ... */
    exit 0; // 0 Sucesso, 1 Falha
}

int main()
{
    funcaoQualquer();
}
```