INTRO-CS-3 - Projeto de algoritmo e solução de problemas

1. Operações
2. Estruturas de controle

1. Operações

Aritméticas

Todos os operadores aritméticos são binários. Eles já foram discutidos brevemente na seção de Operadores de atribuição. Cinco operadores representam as quatro operações básicas da matemática e a operação de módulo, que retorna o resto da divisão do primeiro operando pelo segundo. Veja exemplos de cada uma dessas operações:

- **Adição:** x = 5 + 6
- **Subtração:** x = 7 - 4
- **Multiplicação:** x = 5 * 3
- **Divisão:** x = 10 / 5
- **Módulo:** x = 12 % 5

Algumas linguagens suportam operadores aritméticos adicionais, como por exemplo, de exponenciação. Geralmente, é usado um ^ para essa operação.

Precedência de operadores aritméticos:

- **Sinal** (): se existirem parênteses aninhados, os mais internos são avaliados primeiro
- **Sinal** *, / e %
- **Sinal** + e -

Operadores adicionais:

- **Incremento** ++
- **Decremento** - -

```
programa {
    funcao inicio() {

        // Operações aritiméticas
        escreva(5 + 6, "\n")
        escreva(7 - 4, "\n")
        escreva(5 * 3, "\n")
        escreva(10 / 5, "\n")
        escreva(12 % 5, "\n\n")

        // Precedência de operadores aritiméticos
        inteiro x
        x = (5 * 4) / 2 - 1
        escreva("Valor de x: ", x, "\n\n")

        // Operadores adcionais
```

```
        inteiro y = 5
        escreva(y++, "\n")
        escreva(y--, "\n")
    }
}
```

Relacionais

Operadores relacionais só podem ter como resultado um valor booleano, que pode assumir apenas os valores verdadeiro ou falso. Os operadores de comparação mais comuns (todos binários), que estão presentes na maioria das linguagens de programação, são:

- **igual:** ==
- **diferente:** !=
- **maior que:** >
- **menor que:** <
- **maior ou igual a:** >=
- **menor ou igual a:** <=

Os símbolos acima podem variar de acordo com as linguagens de programação. E é muito comum que em pseudocódigos e fluxogramas sejam usados outros símbolos para os operadores de igualdade e diferença. No caso do operador de igualdade, é comum usar apenas um =. No caso do operador de diferença, algumas variações são o <> e o =/=.

```
programa {
    funcao inicio() {
        inteiro x = 2

        escreva(x == 2, "\n")
        escreva(x != 2, "\n")
        escreva(x > 2, "\n")
        escreva(x < 2, "\n")
        escreva(x >= 2, "\n")
        escreva(x <= 2, "\n")
    }
}
```

Lógicos

Operadores lógicos servem para conectar duas expressões ou para negar uma expressão. Os operadores lógicos mais usados são:

- **e**: O operador e (and nas linguagens de programação) serve para conectar duas expressões, de modo que as duas precisam ser verdadeiras para que a expressão seja verdadeira.
- **ou**: O operador ou também é binário assim como o e, só que ele precisa que pelo menos uma das duas expressões que ele liga seja verdadeira, e não que ambas sejam necessariamente verdadeiras, assim como o e.
- **nao**: Ao contrário dos operadores e e ou, o nao é um operador unário, o que quer dizer que ele contém apenas um operando. Ele é muito simples: ele serve apenas para negar uma expressão, invertendo o valor lógico dela.

- **xor**: É bem parecido com o ou. A única diferença é que pra que o resultado dele seja verdadeiro, só uma das expressões deve ser verdadeira. Na prática, ele é verdadeiro quando uma das expressões é verdadeira e a outra é falsa, pois ele precisa que pelo menos uma seja verdadeira, mas não ambas. Ele não é suportado no Portugol.

```
programa {
    funcao inicio() {

        // e
        escreva(verdadeiro e verdadeiro, "\n")
        escreva(falso e falso, "\n")
        escreva(verdadeiro e falso, "\n")
        escreva(falso e verdadeiro, "\n\n")

        // ou
        escreva(verdadeiro ou verdadeiro, "\n")
        escreva(falso ou falso, "\n")
        escreva(verdadeiro ou falso, "\n")
        escreva(falso ou verdadeiro, "\n\n")

        // nao
        escreva(nao(verdadeiro e verdadeiro), "\n")
        escreva(nao(verdadeiro ou verdadeiro), "\n\n")
    }
}
```

Caracteres

Vou falar sobre o operador de caractere +. Ele é um operador de concatenação de strings (que são cadeias de caracteres), quando é usado com dois valores (variáveis ou constantes) que são um caractere ou uma cadeia de caracteres. Em algumas linguagens, esse operador é representado por outro símbolo. Por exemplo, no PHP, ele é representado por um ponto. O Portugol suporta o operador de concatenação.

```
cadeia nome = "Lorraine"
cadeia sobrenome = "Santos"
cadeia nomeCompleto = nome + " " + sobrenome
escreva(nomeCompleto)
```

2. Estruturas de controle

No meio dos anos 60, matemáticos provaram que qualquer programa, não importa o quão complicado ele seja, pode ser construído usando uma ou mais de apenas três estruturas, que são: sequência, seleção e iteração.

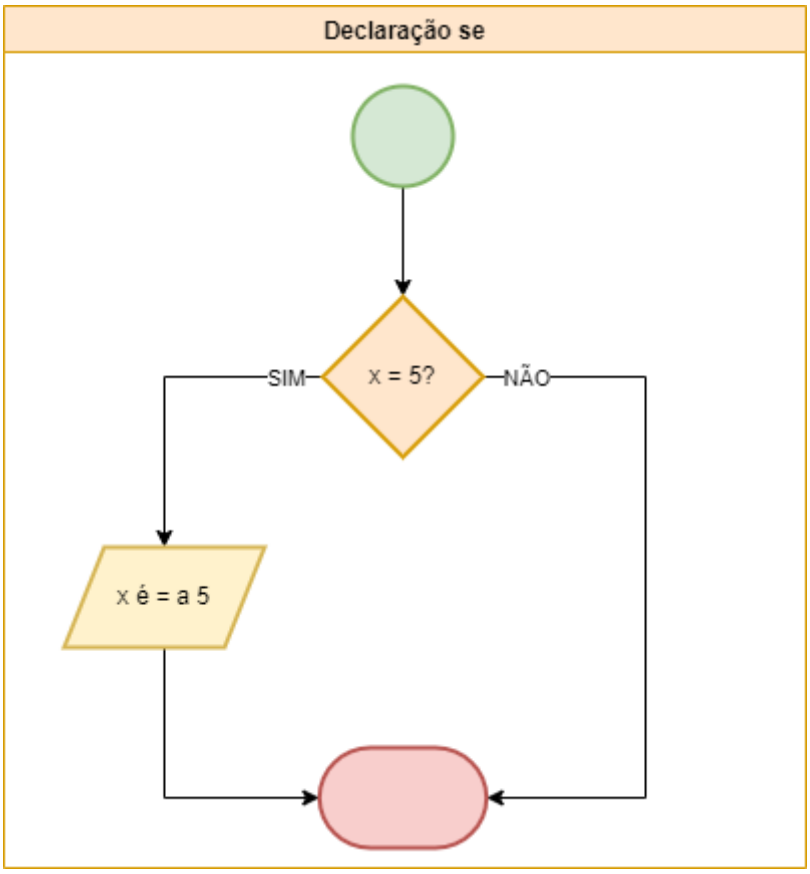
Na sequência, as ações são feitas uma após a outra e o fluxo do programa é linear. Uma vez que você começa uma série de ações em uma sequência, você deve continuar passo-a-passo até que a sequência termine.

Estrutura se

Ela simplesmente recebe uma condição entre parênteses, e se ela for verdadeira, o código que está dentro das chaves é executado. A sintaxe dela no Portugol é bem parecida com a sintaxe usada pelas linguagens de programação:

```
inteiro x = 5
se (x == 5) {
    escreva("x é igual a 5")
}
```

Fluxograma:

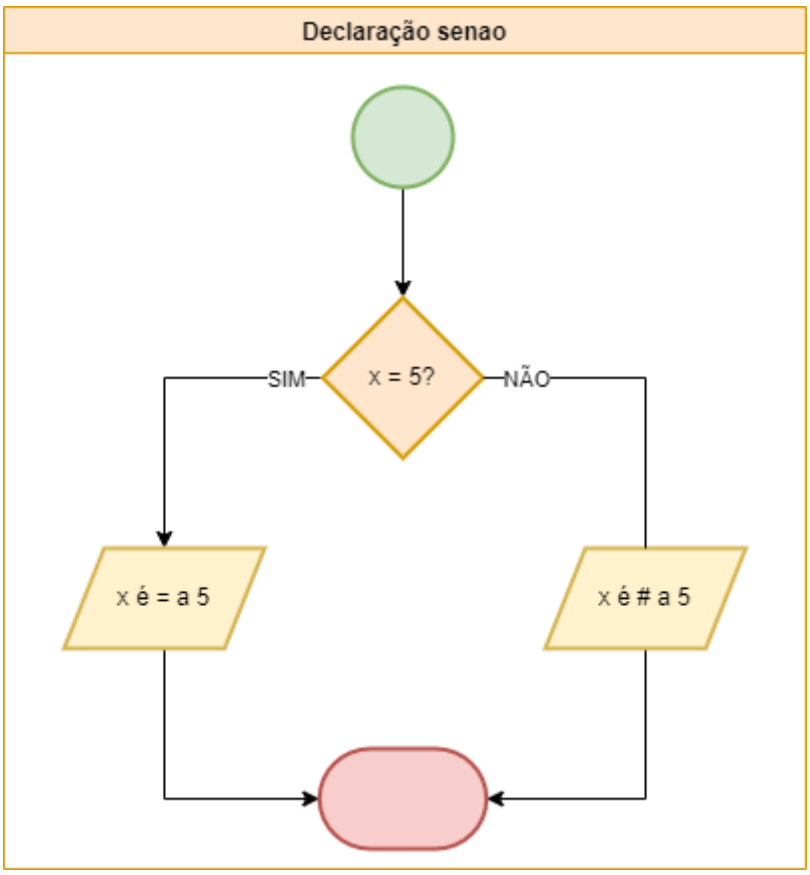


Estrutura senao

Ainda no exemplo anterior, o que faríamos se quiséssemos mostrar uma mensagem caso a variável x não fosse igual a 5? Para isso, existe uma estrutura de controle chamada senao. É o else das linguagens de programação. Ela deve ser colocada imediatamente depois de um se ou de um senao se, que você vai ver daqui a pouco. Veja como ficaria o código com o senao:

```
inteiro x = 5
se (x == 5) {
    escreva("x é igual a 5")
}
senao {
    escreva("x é diferente de 5")
}
```

Fluxograma:

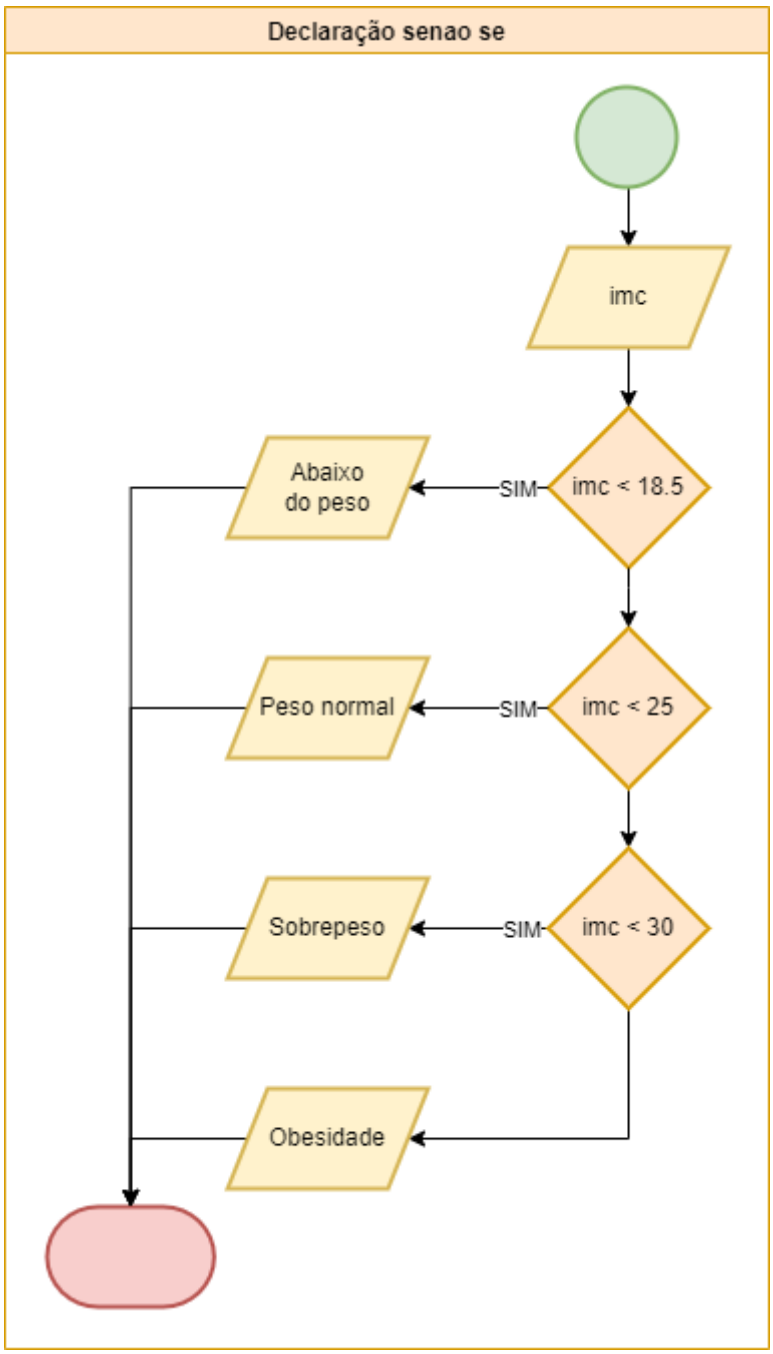


Estrutura senao se

Essa estrutura precisa vir imediatamente após um se ou um senao se. Ela é executada quando a estrutura anterior não tem o seu código executado. Até esse ponto ela é igual ao senao. A diferença é que nem sempre o senao se será executado, porque ele tem uma condição. Se ela for verdadeira, ele é executado. Nesse ponto, ele é igual ao se. Então, podemos dizer que essa estrutura pega um pouco das duas estruturas. Inclusive, o nome dela indica isso.

```
real imc
leia(imc)
se (imc < 18.5) {
    escreva("Abaixo do peso")
}
senao se (imc < 25) {
    escreva("Peso normal")
}
senao se (imc < 30) {
    escreva("Sobrepeso")
}
senao {
    escreva("Obesidade")
}
```

Fluxograma:

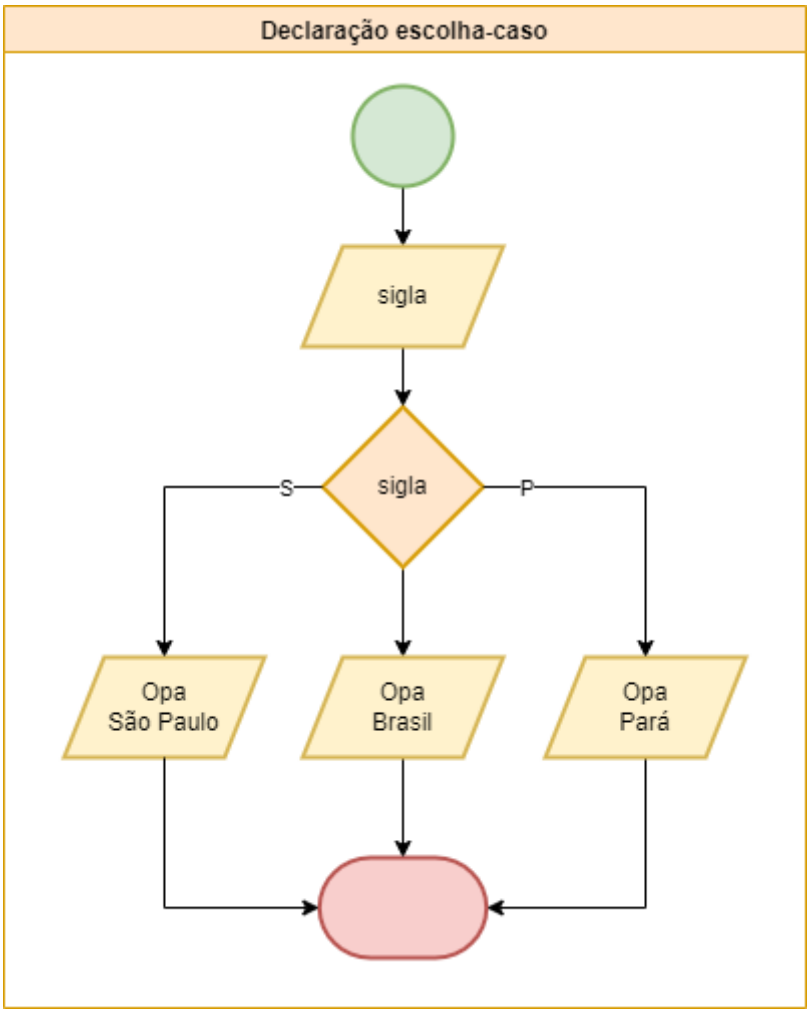


Estrutura escolha-caso

Essa estrutura é equivalente ao switch das linguagens de programação. Nessa estrutura, você pega uma variável qualquer do tipo inteiro ou caracter, e cria ramificações para os valores que a variável pode ter. Exemplo:

```
caracter sigla
leia(sigla)
escolha (sigla) {
    caso 'S':
        escreva("Opa São Paulo")
        pare
    caso 'P':
        escreva("Opa Pará")
        pare
    caso contrario:
        escreva("Opa Brasil")
}
```

Fluxograma:



Para refinar seu conteúdo acesse a documentação do Portugol, segue o Link da Documentação.