INTRO-CS-4 - Concepção de algoritmos e solução de problemas - Avançado

1. Array
2. Estruturas de repetição
3. Algoritmos de classificação
4. Teste de mesa

1. Array

Uma array é um tipo de dado que armazena vários valores. Exemplos: uma lista de fornecedores de uma empresa ou uma lista com o número de vendas de um determinado produto em cada mês. Em algumas linguagens, esses valores precisam ser do mesmo tipo, como C++, e em outras, esses valores podem ser de tipos diferentes, como PHP.

Vetor

Uma array pode ter várias dimensões. Uma array unidimensional é chamada de vetor. segue exemplo de declaração de vetor:

```
inteiro fotos[10]
inteiro vitorias_equipes[] = {3, 1, 2, 1, 1}
real precos_produtos[3] = {10.30, 3.45, 2.10}
```

No Portugol Studio, um vetor é declarado da seguinte forma: primeiro, informa-se o tipo da variável, seguido pelo identificador dela. Depois, coloca-se colchetes, que é o que indica que a variável é uma array. Se você quiser indicar o tamanho do vetor, basta informá-lo nos colchetes. O tamanho do vetor é obrigatório se os elementos do vetor não forem inicializados.

Se os elementos do vetor forem informados, o tamanho se torna opcional, porque ele pode ser inferido. Quando os elementos do vetor são definidos, eles devem ser colocados entre chaves e separados por vírgulas.

Matrizes

Arrays podem ter várias dimensões. Uma array unidimensional é chamada de vetor. Já uma array bidimensional é chamada de matriz:

```
//Matriz que armazena altura e peso
real dados_pessoas[3][2]
real dados_pessoas[3][2] = {{1.77, 78.0}, {1.54, 52.0}, {1.65, 57.0}}
real dados_pessoas[][] = {{1.77, 78.0}, {1.54, 52.0}, {1.65, 57.0}}
```

Veja que a declaração de uma matriz segue a mesma lógica da declaração de um vetor. Basicamente, se adiciona um colchete a mais, para indicar a segunda dimensão. E, na definição dos elementos da matriz, cada dimensão é definida dentro de suas próprias chaves. E as chaves são separadas por vírgulas. São chaves dentro de chaves. Note também que os tamanhos das dimensões da matriz podem ser inferidos, como acontece na terceira declaração.

índices

Depois que uma array foi declarada e você definiu os elementos dela, você pode usar os elementos da array da mesma forma que você usuaria variáveis simples. Ou seja, você pode atribuir valores a elementos da array, imprimi-las, etc. Cada elemento de uma array é diferenciado dos outros por um índice único, que é usado para acessar o elemento da array. Na maioria das linguagens, o índice é obrigatoriamente numérico. Em linguagens onde o índice deve ser numérico, ele quase sempre indica a posição de um item particular dentro de uma matriz, pois os itens quase sempre começam do 0 e aumentam de 1 em 1.

```
inteiro vitorias_equipes[] = {3, 1, 2, 1, 1}
real dados_pessoas[3][2] = {{1.77, 78.0}, {1.54, 52.0}, {1.65, 57.0}}
escreva(vitorias_equipes[0])
vitorias_equipes[1] = 2
escreva(vitorias_equipes[1])
dados_pessoas[1][0] = 1.52
escreva(dados_pessoas[1][0])
```

Perceba como a sintaxe para definir e para acessar o valor de um elemento de uma array é a mesma. Se for um vetor, apenas informe o nome da array e coloque o índice entre colchetes. Se for uma matriz, adicione o índice da segunda dimensão também.

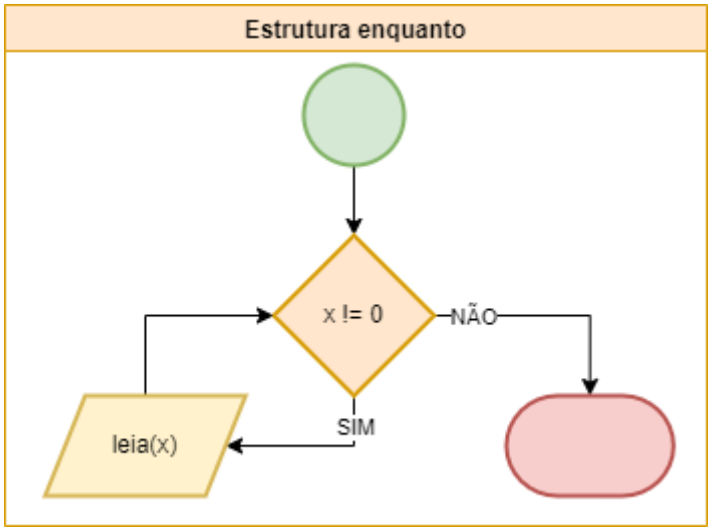
2. Estruturas de repetição

Essa estrutura repete ações enquanto uma condição permanecer verdadeira. Essa condição pode ser tanto uma condição simples, como também uma condição composta, que nada mais é do que um grupo de condições. Cada ciclo do loop é chamado de iteração.

Estrutura enquanto

Um tipo de loop muito comum é o enquanto, que é o while das linguagens de programação. Veja abaixo um algoritmo:

Fluxograma:



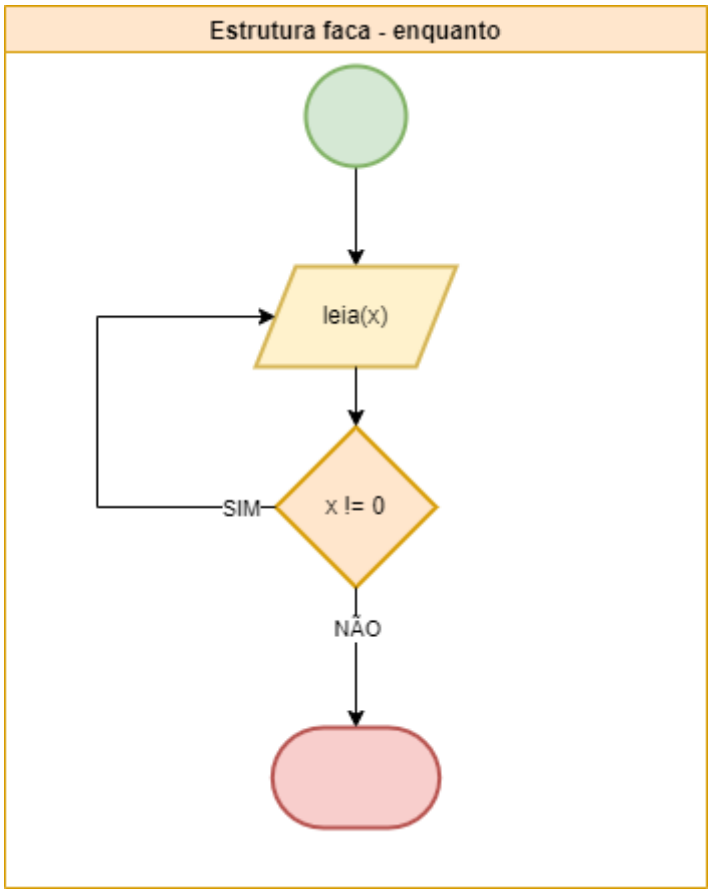
```
inteiro x = 1
enquanto (x != 0) {
    leia(x)
}
```

Antes da primeira iteração, as condições do loop são avaliadas. Por isso, esse tipo de loop é chamado de repetição pré-testada. Por isso, a variável numero recebeu o valor 1. Se ela tivesse recebido o valor 0, o programa nem entraria no loop. O loop continua sendo executado enquanto a sua condição permanecer verdadeira, ou seja, enquanto o valor da variável numero for diferente de 0. A cada iteração, todas as instruções do loop são executadas.

Estrutura faca - enquanto

Um outro tipo de loop é o do-while (faz-enquanto). Nesse tipo de loop, pelo menos uma iteração ocorre, porque a avaliação da condição se dá após a execução da iteração. Por isso, esse tipo de repetição é chamada de repetição pós-testada:

Fluxograma:




```
inteiro x
faca {
    leia(x)
} enquanto (x != 0)
```

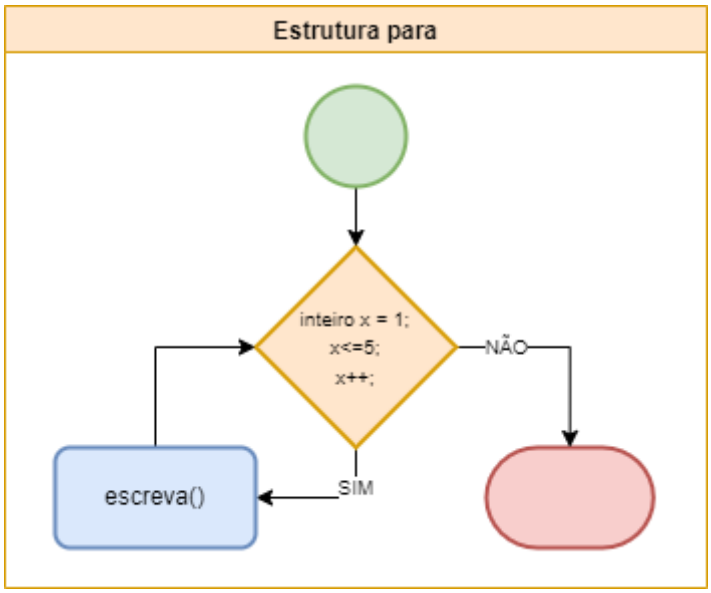
Note que aqui não é necessário atribuir um valor inicial para a variável x que seja diferente de 0, porque a primeira iteração está garantida. Por isso, esse algoritmo ficou mais adequado a esse tipo de situação, até porque evitou a necessidade da atribuição de um valor inicial à variável usada na condição.

Estrutura para

A maioria das linguagens de programação suportam o for. Ele define três ações em uma sintaxe compacta. Essas ações são inicialização, condição e atualização. A inicialização é usada para inicializar variáveis e é a primeira coisa que é feita no for. Só é feita no começo. A condição é feita antes de cada iteração, inclusive a primeira. A atualização é feita após cada iteração.

O uso mais comum de um for é inicializar uma variável, que é chamada de variável de controle, testa-lá na condição do for e depois atualiza-lá. Por causa do uso da variável de controle, esse tipo de repetição é chamado também de repetição com variável de controle.

Fluxograma:




```
para (inteiro x = 1; x <= 5; x++) {
    escreva(x, "\n")
}
```

Não é possível omitir a condição no Portugol Studio, embora algumas linguagens de programação, como o PHP, permitam a omissão da condição.

Pare e continue

Duas instruções muito comuns em linguagens de programação são o pare (que é o break das linguagens de programação) e o continue. Elas são usadas em estruturas de repetição, então eu posso usar em qualquer uma das estruturas de repetição que eu mostrei aqui. Além disso, o pare também é usado na estrutura de decisão escolha-caso. O continue não é suportado no Portugol.

```
para (inteiro x = 10; x >= 1; x--) {
    escreva(x, "\n")
    se (x == 5) {
        pare
    }
}
```

3. Algoritmos de classificação

A classificação ordena uma lista. É possível classificar o algoritmo de ordenação pela estabilidade. Para uma breve explicação a ordenação é classificada em estável e instável.

Dado o seguinte cenário de lista:

(1, 2) (9, 7) (3, 4) (8, 6) (9, 3)

- Em um cenário estável, a ordenação ficaria: **(1, 2) (3, 4) (8, 6) (9, 7) (9, 3)**
- Em um cenário instável, a ordenação ficaria: **(1, 2) (3, 4) (8, 6) (9, 3) (9, 7)**

Tipos de classificação estáveis, mais comuns:

- Merge sort;
- Insertion sort;
- Radix sort;
- Tim sort;
- Bubble Sort.

Tipos de classificação instáveis, mais comuns:

- Heap sort;
- Quick sort.

Cada um dos algoritmos de ordenação possuem suas complexidade e diversas maneiras para se implementar, para uma visualização geral sobre o funcionamento de ordenação veja o video Click aqui

4. Teste de mesa

Teste de mesa é uma simulação da execução de um programa de forma manual, geralmente feita no papel. Não há regras rígidas para criar um teste de mesa, mas geralmente ele é feito de uma de duas formas. Considere o pseudocódigo abaixo:

```
inteiro a, b, soma, diferenca
escreva("Digite dois números inteiros\n")
leia(a, b)
soma = a + b
diferenca = a - b
escreva("A soma dos dois números é ", soma, "\n")
escreva("A diferença dos dois números é ", diferenca, "\n")
```

a	b	soma	diferenca
4	3	7	1
20	10	30	10
10	15	25	-5

Testes de mesa são mais usados para propósitos didáticos, ou quando não se dispõe de um computador enquanto se está criando um algoritmo e deseja-se testar o algoritmo, geralmente com valores de input diferentes. São usados também quando se tem dificuldade para entender o funcionamento de um algoritmo. Então, fazendo o teste de mesa, fica mais fácil entender o que o algoritmo faz.

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