Dicas para codificar um fluxograma

Uma etapa importante para o desenvolvimento de programas e algoritmos é a construção de fluxogramas. Com esse tipo de diagramas, podemos visualizar a sequência de ações que nosso computador irá executar. No entanto, pode ser difícil traduzir esses diagramas para a forma textual, ou seja, o código em uma linguagem de programação.

Este documento busca ajudar esse processo de tradução, mostrando alguns exemplos de estruturas comuns em fluxogramas e sua tradução em código, assim como algumas observações. Todos os códigos estão em **JavaScript**, menos as condições / operações, que estão simplificadas.

Prof. Enric Llagostera - enricllagostera@gmail.com

Entrada e saída de dados

Tanto a entrada quanto a saída (no sentido de exibição para o usuário) de dados tem seu próprio bloco especial em fluxogramas. Aqui estão eles com seus respectivos códigos em JavaScript (lembrando que `prompt()` sempre retorna um valor string / texto, que deve ser convertido em outro tipo se você quer números, por exemplo).


```
var meuNome = prompt();
```

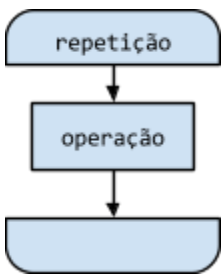

```
console.log (meuNome);
```

Estruturas de repetição

Nos fluxogramas, as estruturas de repetição são representadas por um bloco de abertura e um bloco de fechamento.

`for`

Particularmente útil quando três informações são conhecidas: condições iniciais das variáveis de controle, condição para a repetição e a mudança que as variáveis de controle irão sofrer a cada passo da repetição. É a melhor forma de repetição para situações em que a quantidade de repetições é essencial, como na hora de percorrer um vetor. Esta estrutura aplica a mudança definida nas variáveis de controle automaticamente.




```
for (var ind = 0; ind < 5; ind++) {
      // comandos
}
```

`while` e `do - while`

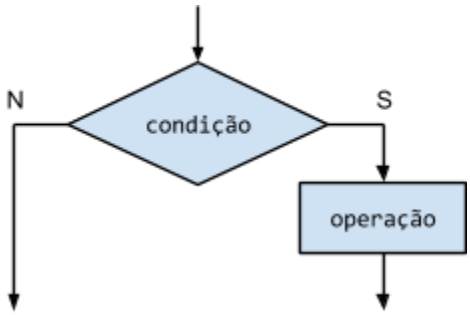
Estas estruturas de repetição são bastante úteis para momentos em que o número de vezes que a repetição vai acontecer é desconhecida ou muito imprevisível. A principal diferença entre as duas é o momento em que checam sua condição de repetição: `while ()` verifica antes de executar os comandos, enquanto que `do {} while ()` executa os comandos uma vez antes de checar a condição de repetição.

```
while (condição) {
    // comandos
}
```

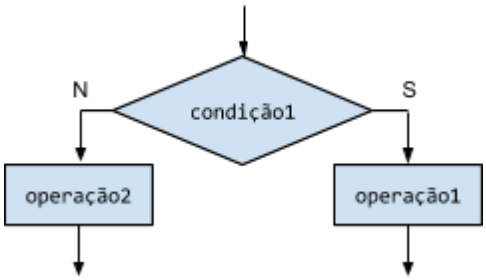
```
do {
   // comandos
} while (condição);
```

Estruturas de decisão: if / if - else / if - else if - else

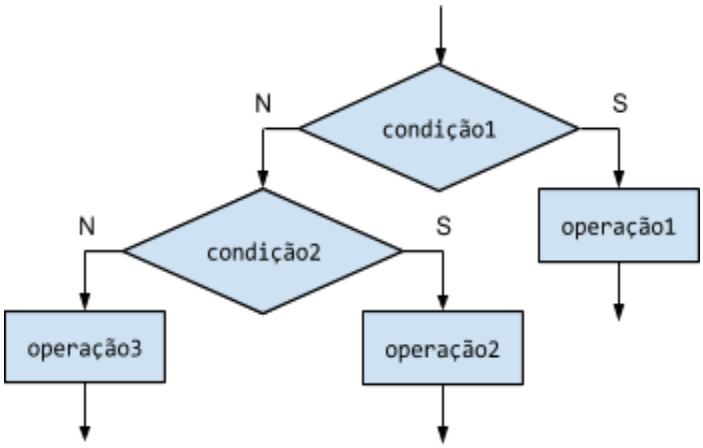
Estas são as estruturas de decisão mais comuns. O **if** é usado quando apenas o lado verdadeiro de uma condição tem operações específicas. O **if - else** nos permite definir operações específicas tanto para condições verdadeiras quanto para sua negação. O **if - else if -else** é útil quando temos operações específicas no caso de uma condição verdadeira e sua negação tem como primeira operação uma nova condição. É importante lembrar que essas estruturas podem ser combinadas e colocadas em sequência.




```
if (condição) {
    operação;
}
```



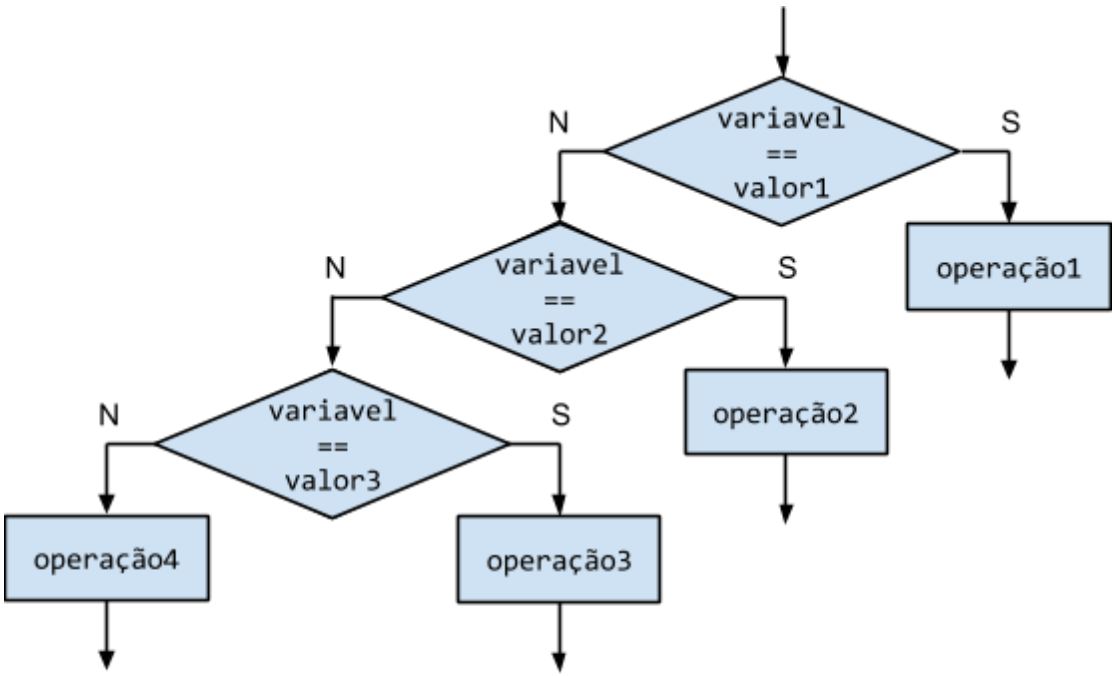

```
if (condição1) {
    operação1;
}
else {
    operação2;
}
```




```
if (condição1) {
     operação1;
}
else if (condição2) {
     operação2;
}
else {
     operação3;
}
```

Estruturas de decisão: switch - case

Esta estrutura depende de que as condições nas decisões sejam relacionadas a uma mesma variável. Também é particularmente útil quando muitas alternativas mutuamente exclusivas são necessárias. No entanto, é importante lembrar que a linguagem JavaScript não permite condições relacionais (maior-igual, maior, menor, menor-igual) dentro de um switch-case. Exemplos: inputs, múltipla escolhas, casos com valores bem definidos.

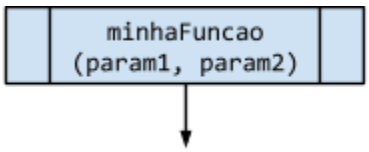



```
switch (variavel) {
    case valor1 : operação1; break;
    case valor2 : operação2; break;
    case valor3 : operação2; break;
    default : operação4; break;
}
```

Funções

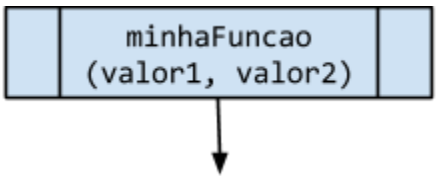
Em fluxogramas, representamos funções com um tipo especial de bloco. Esse bloco pode aparecer em dois momentos: quando de sua **declaração** e quando a função é **chamada** para ser executada. Na **declaração**, o bloco mostra o nome da função e os parâmetros que deve receber, com os nomes que esses parâmetro terão dentro da função. Na **chamada**, o bloco contém o nome da função e os valores sendo passados como parâmetros.

Declaração de função




```
function minhaFuncao (param1, param2)
{
    // comandos
}
```

Chamada de função




```
minhaFuncao (valor1, valor2);
```