Estruturas de Dados - Revisão de Fundamentos de Programação 01

Revisão de fundamentos de programação

- Importância de Fundamentos de Programação
  - Passar conhecimentos básicos de programação
    - tipos de variáveis, vetores, matrizes, loops, ...
  - Introduzir a utilização desses conhecimentos em algoritmos
- Resultado:
  - maior facilidade para resolver problemas
  - capacidade de identificar maneiras diferentes de resolver um mesmo problema (e alguma capacidade de identificar a mais eficiente)
  - menor dificuldade para aprender uma nova linguagem

Revisão de fundamentos de programação

- Linguagem de programação?
- Revisão de fundamentos e apresentação de algoritmos em sala: **pseudocódigo**
  - simples, de fácil entendimento
  - sem preocupação com os termos reservados de uma linguagem ou com a semântica aplicada
  - "programar" sem preocupação com erros sintáticos (ex: ponto e vírgula, parênteses, ...)
  - foco é 100% voltado a como resolver o problema
  - fácil adaptação do código para linguagens de programação
- Programação: **C**
  - linguagem com sintaxe simples e com boa portabilidade
  - capaz de resolver os problemas que nos interessam

Revisão de fundamentos de programação

- Variável x Constante:
  - ambos representam espaços reservados na memória
  - variável: valor pode variar durante a execução
    - variável iteradora em um loop
    - valor de um somatório
    - ⋮
  - constante: valor não muda com o tempo
    - $\pi = 3,141592\ldots$
    - $e = 2,718281\ldots$
    - ⋮

Revisão de fundamentos de programação

- Tipos de variáveis:
  - inteiro
    - armazena somente números inteiros (sem casas decimais)
  - real
    - armazena números reais (permite casas decimais)
  - caractere
    - armazena um caractere (letra, dígito, caractere especial)
  - booleano
    - armazena um valor verdadeiro ou falso
    - obs: C não tem variável booleana, portanto utilizamos 0 e 1 para representar verdadeiro e falso, respectivamente

Revisão de fundamentos de programação

- Dados estruturados:
  - vetor
    - armazena uma sequência unidimensional de variáveis de um mesmo tipo
  - matriz
    - armazena uma sequência multidimensional de variáveis do mesmo tipo
  - registro
    - armazena um conjunto de variáveis que podem ser de tipos diferentes

Revisão de fundamentos de programação

- Condicional:
  - verifica uma dada condição e caso ela seja satisfeita, executa um bloco de código uma única vez
  - se (*condição*) então {*bloco de código*}
    - a condição a ser verificada deve ser booleana (verdadeira ou falsa)
  - podemos adicionar um bloco de código adicional que será executado caso a condição não seja satisfeita
    - se (*condição*) então {*bloco de código 1*} senão {*bloco de código 2*}
  - também podemos aninhar condicionais
    - se (*condição 1*) então {*bloco de código 1*} senão se (*condição 2*) então {*bloco de código 2*}
    - se (*condição 1*) então {se (*condição 2*) então {*bloco de código*}}

Revisão de fundamentos de programação

- Loops:
  - executam um bloco de código várias vezes, até que uma certa condição não seja mais satisfeita
  - para ($condição + passo$) faça $\{bloco\ de\ código\}$
    - sua própria sintaxe já estabelece o passo (crescimento de uma variável iteradora) utilizado na condição
    - no para, o ideal é não modificarmos o valor do contador dentro do bloco de código
    - obs: para simplificar a notação, quando o passo for igual a 1 não precisamos especificá-lo

Revisão de fundamentos de programação

- Loops:
  - enquanto (*condição*) faça {*bloco de código*}
    - verifica a condição antes de executar o bloco de código
    - sua sintaxe contém apenas a condição a ser verificada
    - logo, ele exige que a variação do contador seja expressa dentro do loop, no bloco de código
  - repita {*bloco de código*} enquanto (*condição*)
    - verifica a condição após executar o bloco de código uma vez
    - funcionamento semelhante ao enquanto, mas o bloco de código sempre é executado pelo menos uma vez

Revisão de fundamentos de programação

Exemplo de pseudocódigo utilizando **para**:

Algoritmo: Fatorial(n)

Entrada: natural n

Saída: valor de $n!$

```
1 f = 1
2 para i = 2 até n faça
3 |   f = f · i
4 retorne f
```

Revisão de fundamentos de programação

Exemplo de pseudocódigo utilizando **enquanto**:

Algoritmo: Fatorial(n)

Entrada: natural n

Saída: valor de $n!$

```
1 f = 1
2 enquanto n > 1 faça
3 |   n = n − 1
4 |   f = f · n
5 retorne f
```

Revisão de fundamentos de programação

Exemplo de pseudocódigo utilizando **enquanto** (incorreto):

Algoritmo: Fatorial(n)

Entrada: natural n

Saída: valor de $n!$

```
1 f = 1
2 enquanto n > 1 faça
3 |   n = n − 1
4 |   f = f · n
5 retorne f
```

Revisão de fundamentos de programação

Exemplo de pseudocódigo utilizando **enquanto** (corrigido):

Algoritmo: Fatorial(n)

Entrada: natural n

Saída: valor de $n!$

```
1 f = 1
2 enquanto n > 1 faça
3 |   f = f · n
4 |   n = n − 1
5 retorne f
```

Revisão de fundamentos de programação

Exemplo de pseudocódigo utilizando **repita**:

Algoritmo: Fatorial(n)

Entrada: natural n

Saída: valor de $n!$

```
1 f = 1
2 repita
3 |   f = f · n
4 |   n = n − 1
5 enquanto n > 1
6 retorne f
```

Revisão de fundamentos de programação

No loop repita, o bloco de código é executado antes de se verificar a condição

Para $n = 0$, o retorno da função é 0, mas $0! = 1$

Exemplo de pseudocódigo utilizando **repita** (incorreto):

Algoritmo: Fatorial(n)

Entrada: natural n

Saída: valor de $n!$

```
1 f = 1
2 repita
3 |   f = f · n
4 |   n = n − 1
5 enquanto n > 1
6 retorne f
```

Exemplo de pseudocódigo utilizando **repita** (corrigido):

Algoritmo: Fatorial(n)

Entrada: natural n
Saída: valor de $n!$

```
1 se n == 0 então
2 |   retorne 1
3 f = 1
4 repita
5 |   f = f · n
6 |   n = n − 1
7 enquanto n > 1
8 retorne f
```