Universidade de Brasília

Nome: Waliff Cordeiro Bandeira

Matrícula: 17/0115810

# Métodos de Programação - 2/2018

Laudo – Trabalho 3

Neste laudo avaliaremos o quanto conseguimos seguir o checklist de “como fazer o código”

## Checklist

## Como fazer o código

1. **Valores negativos:** Quando sabemos que o valor negativo não é uma opção, podemos garantir através do tipo de variável que isso não aconteça trocando, por exemplo:

   **int idade; por unsigned int idade;**

2. **Valores constantes:** Na situação em que temos certeza de que um valor é constante, podemos defini-lo de tal forma, por exemplo:

   **int dias_da_semana = 12;**

   **pode ser substituído por const unsigned dias_da_semana = 12;**

   Ou seja, devemos substituir as constantes por nomes de fácil entendimento.

3. **Declaração de arrays:** Ao declarar um array devemos deixar de fácil entendimento o que o tamanho dele significa, por exemplo, na situação int array[13]; não é trivial saber o que significa essa constante = 13, mas se trocarmos por uma expressão int array[meses+1]; fica claro que o array cabe a quantidade total de meses e mesmo que uma pessoa vá ler o código anos depois, ela conseguirá instantaneamente identificar o que significa.

4. **Uso dos parênteses:** No momento em que a expressão está sendo escrita pode parecer algo trivial, mas os parênteses não devem ser omitidos de suas expressões, por exemplo:

   **if ( a>b && b<=c || b > d),** ao escrever algo desse tipo pode parecer bem simples a interpretação, mas para evitar confusões, podemos deixar de forma muito clara com a utilização dos parênteses:

   **if ( ((a>b) && (b<=c)) || (b > d))**

   O mesmo serve para equações matemáticas area_trapezio = ((B+b) x h)/2, nesse caso o uso indevido dos parênteses pode até mesmo ocasionar em erros, mas devemos sempre lembrar do uso adequado dos parênteses para o bom entendimento do código.

5. **Identação:** A identação nada mais é do que manter a organização de seu código, existem linguagens que essa identação é feita até de forma obrigatória, como o python, sendo que se o programa não estiver identado de forma correta o seu programa não será executado da forma que se espera, essa é uma boa prática para seu próprio entendimento sobre o que está acontecendo e o entendimento de outras pessoas que porventura irão ler o seu código. A seguir temos o exemplo de um código não identado:

```
package br.com.javamagazine;
  class Vendas implements InterfaceVendas{
  public String nomeCliente;
  protected String descProduto;
  private double valor=50;
  public double altera_valor(double valor){
  this.valor = valor;
  return valor;
  }
  public void imprime(double valor){
  if (valor>100)
  JOptionPane.showMessageDialog(null,"Valor acima do permitido");
  else JOptionPane.showMessageDialog(null,"Valor="+valor);
  }
  }
```

   E um código identado:

```
package br.com.javamagazine;

 import javax.swing.JOptionPane;

 class Vendas implements InterfaceVendas {

   public String nomeCliente;
   protected String descProduto;
   private double valor = 50;

   public double alteraValor(double valor){
     this.valor = valor;
     return valor;
   }

   public void imprimeValor(double valor){
     if (valor > 100)
       JOptionPane.showMessageDialog(null,"Valor acima do permitido");
     else
       JOptionPane.showMessageDialog(null,"Valor="+valor);
   }

 }
```

Podemos perceber que o primeiro exemplo vem de forma inadequada e o segundo de forma mais legível e de fácil entendimento, podendo assim perceber o que é ou não uma boa prática a ser seguida.

6. **Nomes intuitivos para variáveis:** Prefira usar nomes de fácil entendimento, assim como na situação de valores constantes isso serve para qualquer valor do seu código no qual essa prática facilite o entendimento, por exemplo:

   Nome aleatório: char Array[100] = “Waliff”;

   Nome intuitivo: char ArrayNome[10] = “Waliff”;

7. **Comparação com ponto flutuante:** Não se deve comparar uma variável com igualdades de ponto flutuante, por exemplo:

   if (variável_float == 1.7)

   É preferível usar <e> dentro de uma região de aproximação.

8. **Fazer comentários relevantes:** A melhor opção, sem dúvidas, é a prática de variáveis e métodos bem nomeados, se o código puder ser entendido sem a adição de comentários fica na sua situação ideal, porém existem casos nos quais não conseguimos deixa-lo totalmente legível sem a adição de comentários. Sendo assim, comentários que consigam detalhar o que está sendo proposto, seus

objetivos e parâmetros que foram mal especificados é de extrema importância para o bom entendimento e auxilio nas futuras manutenções que serão prestadas no código.

9. **Redundância no código:** Ao perceber que uma parte do código será escrita mais de uma vez, devemos pensar em uma maneira de evitar tal prática, como por exemplo o uso de funções:

   No código:
   ```
   int Matriz[7][7];
   int Matriz2[5][5];
   ```

   Não fazer :
   ```
   for(int i; i <7; i++){
           for(int j; i < 7; j++){
                   Matriz[i][j] = 100;
           }
   }
   for(int i; i <5; i++){
           for(int j; i < 5; j++){
                   Matriz2[i][j] = 99;
           }
   }
   ```

   Podemos evitar tal redundância com a criação de uma função:

   ```
   void incializa_matriz(**matriz, int linhas, int colunas, int valor){
           for(int i; i <linhas; i++){
                   for(int j; i < colunas; j++){
                           matriz[i][j] = valor;
                   }
           }
   }
   incializa_matriz(Matriz, 7, 7, 100);
   incializa_matriz(Matriz, 5, 5, 99);
   ```

10. E**vitar notação húngara:** Evitar o uso de palavras em contextos que não há necessidade, por exemplo:

    ```
    public class Pessoa {
     private String nomeString;
    }
    ```

A palavra String não tem necessidade de ser colocada nessa situação, deixando somente ‘nome’ fica muito mais limpo e possibilita melhor o mesmo nível de entendimento.

11. **Refatoração:** Consiste em deixar o código mais legível possível, retirando coisas inúteis do código e colocando outras que possam nos auxiliar no entendimento, nesse primeiro exemplo:

```
private boolean isStringVazia(String texto){
  if (!StringUtils.isNullOrEmpty(texto) && !texto.equals("")) {
    //...
  }
}
```

Temos o "!texto.equals("")" que não serve para nada, podemos refatorar o código obtendo o mesmo resultado, porém de forma mais simples:

```
private boolean isStringVazia(String texto){
    if (!StringUtils.isNullOrEmpty(texto)) {
      //...
    }
  }
```

Se tivermos receio em algum tipo de refatoração quanto a manutenção da funcionalidade, neste caso um simples teste unitário resolveria rapidamente nossa dúvida e poderíamos seguir tranquilamente em frente. A refatoração é uma das melhores práticas para melhorias no código, ou seja, mesmo que seu código seja eficaz (faça o que se propõe) ele pode ser eficiente (fazer o que se propõe de maneira melhor ainda).

12. **Técnicas para aumentar a eficiência:** Procure sempre que possível usar comandos que possibilitem uma melhor qualidade pras suas funcionalidades, por exemplo, ao invés de usar **w = w + c**, use **w+=c**.
13. **Evite comentários óbvios:** Por exemplo, ao criar uma função booleana o comentário: return true; /*Retorna verdadeiro*/ ou return false; /* Retorna falso*/, são comentários extremamente dispensáveis e devemos evitar esse tipo de comentário, pois ao se comentar o código não temos o intuito de deixa-lo mais poluído e sim mais legível e de bom entendimento.
14. **Percorrer loops somente o necessário:** Continuar percorrendo um vetor, por exemplo, após ter encontrado o que se procurava é perda de processamento desnecessário, isso pode ser resolvido com uma simples verificação se o objeto procurado já foi encontrado, ao ser encontrado podemos dar o comando break; saindo assim do loop e evitando mais repetições desnecessárias.
15. **Nomear funções:** Assim como as constantes e as variáveis, nomear funções é uma tarefa aparentemente simples, mas se soubermos escolher nomes intuitivos

que sejam ligados aos objetivos da função, podemos tornar nosso código mais legível, de forma que muitas vezes o programador nem precise abrir sua função para entender de fato como usá-la. Por exemplo, a função isEmailValid ao ser colocada em um if (isEmailValid(“waliff.cordeiro@gmail.com”)) pode ser lida, se o email é válido, então ... Dessa forma conseguimos obter o resultado esperado, de deixar uma função legível e utilizável sem que mais comentários ou explicações sejam dadas sobre determinada funcionalidade.

## Laudo

1. No programa contador de linhas pôde ser assegurado que não houveram riscos de valores negativos.
2. Não existiram situações em que substituir por um valor constante ajudaria na interpretação, mas o cuidado foi tomado.
3. Os arrays utilizados foram de textos e explicitavam a posição atual e a próxima posição, foi feito no formato que facilita o entendimento.
4. Foram usados os parênteses de forma adequada.
5. O programa foi identado de forma a satisfazer os padrões cpplint.
6. Foram usados nomes intuitivos.
7. Não houve comparação com ponto flutuante.
8. Foram feitos comentários relevantes que colaborassem com o entendimento.
9. Foram evitadas redundâncias.
10. Foi evitada a notação húngara.
11. O programa foi feito utilizando constantes refatorações.
12. Não houveram tantas técnicas de aumento de eficiência.
13. Foram evitados os comentários óbvios.
14. Só foram percorridos os loops necessários.
15. Os nomes dados às funções foram intuitivos.