Capítulo 01 - Documentação e Invariantes

1.1 Documentação

Generiamente documentar é registar alguma informação. Ao escrevermos algoritmos há a nescessidade de documentarmos o que escrevemos, para que posteriormente tanto o próprio desenvolvedor quanto outras pessoas possam compreender o que a aquele trecho de código faz.

Mas o ponto tratado aqui é: qual a forma correta de comentar? A resposta é: Não há! Mas há convenções que foram criadas para dar um norte, uma orientação para que não seja escrito descrições desnescessário em meio ao algoritmo.

Por convenção definimos que a documentação de algoritmo seja realizada atravéz de comentários, respondendo a seguinte pergunta: o que esse trecho de código faz? Note que não é: como esse trecho de código faz, mas sim: o que faz. Entender essa sutil mas enorme diferença lhe guiará na hora de documentar.

Essa pergunta terá que ser repondida TODAS as documentações independente da linguagem do qual está programando.

Podemos usar comentários de linha(// Comentário que usa somente uma linha) e/ou de mais de uma linha(/* Comentário aqui dentro que pode ter quebra de linha*/).

Existem documentações específicas para algumas linguagens de programação, como exemplo o JAVA que em cada método além de responder a pergunta anterior, terá que usar algumas tags(caso seja nescessário) como:
- @author: nome(s) do autor;
- @param : nome e descrição do parâmetro.;
- @return : descrição do valor de retorno;
- @see : referência cruzada;
- @throws: tipo de excessão lancada e as circunstâncias;
- @version) : descrição da versão, segue o exemplos:

```java
/**
 * Cria um objeto e inicializa inicializa os parâmetros: nome, raca, tamanho.
 * @param nome nome do dog
 * @param raca se o dog procria ou não
 * @param tamanho medido em cm, da ponta do nariz a ponta do rabo
 */
public Dog(String nome, String raca, Double tamanho){
    this.nome = nome;
    this.raca = raca;
    this.tamanho = tamanho;
}
```

Declaração de um construtor de Classe em Java.

```c
//A função abaixo recebe uma pilha(p) e um item(i) e insere o item na pilha, retorna 1 caso seja um sucesso a operação.
int pilha_insert(Pilha* p, PilhaItem item){
    elem* elemento;
    elemento = (elem*) malloc(sizeof(elem));

    elemento->prev = p->topo;
    elemento->data = item;

    p->topo = elemento;
    p->tamanho ++;
    return 1;
}
```

Declaração de uma função em C.

Cuidado para não omitir informações ou deixar o comentário vago. Comente sobre todos os parâmetros requeridos, não comente sobre por exemplo as variáveis locais que forem criadas para resolver o problema, deixe de lado também comentários inúteis.

Não listei tudo sobre documentação mas com esse material você já pode iniciar com uma base, note que qualquer dúvida pode usar o google que ele lhe reponderá a qualquer hora do dia ou noite.

1.2 Invaiantes

Invariante é uma condição que vale para início e para o fim de um processo iterativo, ou seja uma condição que não se altera com o decorrer da iteração. Defniição dada no livro: *"Um invaiante é uma relação entre os valores das variáveis que vale para o início de cada iteração do processor iterativo. Os invariantes explicam o funcionamento do processo iterativo e permitem provar, por indução, que ele tem o efeito desejado."*

Mas qual o motivo de tratar sobre invariantes? É simples: descrever invariantes dentro da codumentação enrriquece a mesma, assim quando houver invariantes do seu código, quando for comentar sobre, trate de comentar sobre as invariantes.

"O comentário de uma iteração é provavelmente o único tipo de comentário que vale a pena ser inserirdo no corpo de uma função".

Exemplo:

```c
// A função recebe um inteiro n >=1 e um vetor v e devolve o valor de um elemento máximo de v[0...n-1].
int Max(int v[], int n){
    int j, x = v[0];

    for(j = 1; j < n; v[j])
    {
        if(x < v[j]) x = v[j];
    }
    return n;
}
```

Declaração de uma função em C.

No exemplo acima note que o processo iterativo pelo for tem o invariante: *no início de cada iteração (imediatamente antes da comparção de j com n),*

x é um elemento máximo de v[0… j-1]

O invariante vale para cada início de iteração, por isso ele se chama invariante, por causa que ele não vareia, não se altera ao percorrer da iteração, o que mostra no caso que a função devolve o valor de um elemento máximo de v[0...n-1] .

Com a alteração, descrevendo o invariante temos:

```c
// A função recebe um inteiro n >=1 e um vetor v e devolve o valor de um elemento máximo de v[0...n-1].
int Max(int v[], int n){
    int j, x = v[0];

    // x é um elemento máximo de v[0... j-1]
    for(j = 1; j < n; v[j])
    {
        if(x < v[j]) x = v[j];
    }
    return n;
}
```

Declaração de uma função em C.