Padrão para criação de comentários

Emerson Luiz Navarro Tozette
emr@linux.ime.usp.br
#USP: 2957492 - Graduação

2 de outubro de 2002

1 Objetivo

Como adicionar bons comentários ao código fonte do seu programa.

2 Adicionando comentários ao seu código

Todo programa que se preze possui comentários embutidos em seu código-fonte. Os programadores (principalmente iniciantes) tendem a pensar que os comentários que eles mesmos colocam em seus programas só serão utilizados por outros: os companheiros de sua equipe de trabalho, os programadores futuramente responsáveis pela manutenção do código, o seu chefe de trabalho, ... Na verdade, na maior parte das vezes, quem usa o comentário é o próprio criador do comentário, pois o que você programou hoje você pode não lembrar amanhã. Mas isso não quer dizer que o programador possa criar comentários que sejam ininteligíveis para os outros. É importante que o comentário seja um texto conciso e claro, de leitura rápida, e que forneça uma visão geral a respeito do código para qualquer ser humano interessado em usá-lo ou entendê-lo.

Deve ficar claro que colocar comentários no código não é a mesma coisa que fazer a documentação do seu código. A documentação do código é um texto separado do código, em geral muito mais detalhado e complexo que comentários. O formato da documentação normalmente depende do objetivo da documentação. Uma documentação sobre alguma biblioteca de classes destinada a usuários da biblioteca não precisa fornecer detalhes como, por exemplo, os tipos das variáveis `private` de cada classe. Para o usuário só importa o que a classe faz, a interface de uso da classe, e se a classe é ou não eficiente. Ele não está interessado nos detalhes de implementação. Caso a documentação se destine aos próprios desenvolvedores, incluir detalhes de implementação será importante. No caso dos comentários, como eles ficam junto do código fonte, pode ser importante incluir alguns detalhes de implementação. Por outro lado, incluir detalhes demais no comentário pode poluir muito o código, e caso isso acontecesse seria melhor remover esses detalhes do comentário e colocá-los numa documentação em separado.

Vamos começar discutindo comentários de métodos. Um bom comentário de método deve conter, pelo menos:

- o que o método faz.
- o que o método recebe (i.e., os significados dos seus parâmetros).
- o que o método devolve.

O comentário de um método deve ser um pequeno **manual** do método, que explica o significado dos parâmetros do método, o significado do valor de retorno (se existir), e o que o método faz. Caso os parâmetros não sejam explicados, serei obrigado a ler o código para descobrir os seus significados, e posso demorar até dez vezes mais para aprender a usar a função.

A idéia não é ler o código do método para descobrir o significado dos seus parâmetros, e sim ler o comentário e descobrir isso rapidamente.

Vamos analisar, por exemplo, o seguinte comentário de uma função C chamada `insRN`. Essa função contém uma chamada para outra função, denominada `rota`.

```
// Inclusão em árvore rubro-negra:

int insRN ( int x, Nó * ptv, Nó * ptw, Nó * ptr, int a ) {

           . . .

    rota (ptq, ptv, ptw, ptr, a);

           . . .
}
```

Esse comentário é um excelente exemplo de um antipadrão para criação de comentários. É impossível descobrir o significado dos parâmetros sem ler o corpo da função, tarefa esta que pode ser complicada e demorada. Veja agora um outro comentário para a mesma função:

```
/* Esta função insere o inteiro x em uma árvore rubro-negra. Os ponteiros
ptv, ptw e ptr apontam para o nó corrente da busca, seu pai e seu avô,
respectivamente. A variável a controla as chamadas do procedimento
rota, responsável por manter o equilíbrio da árvore. */
```

Claramente este último comentário é melhor que o anterior, mas ele ainda se refere de forma muito obscura à variável `a`. Um programador encarregado de fazer a manutenção da função (isto é, alguém que obrigatoriamente precisasse ler o código e não apenas utilizá-lo) iria apreciar mais este outro comentário:

```
/*
  Esta função insere o inteiro x em uma árvore rubro-negra.
Os ponteiros ptv, ptw e ptr apontam para o nó corrente da
busca, seu pai e seu avô, respectivamente. A variável a
controla as chamadas do procedimento rota, responsável por
manter o equilíbrio da árvore, e que somente será chamado
se a = 1. Caso a = 2, rota não será mais executado até o
final do processo. Caso a = 0, somente será executada uma
nova chamada de rota quando o nó corrente for avô do atual.
*/
```

É claro que este comentário não contém todos os detalhes de implementação, mas contém detalhes suficientes para que o programador possa começar, sem grandes problemas, a estudar o código. De forma a não poluir muito o código fonte, será melhor incluir os detalhes maiores em uma documentação em separado. Mas para que comentar desse modo o código fonte e ao mesmo tempo ter uma documentação em separado? Porque não fazer apenas a documentação em separado, que é melhor e mais detalhada que comentários? A idéia é que o comentário seja uma *quick-reference* para o código. Como o comentário está inserido no código é muito prático e rápido de usar. E desse modo a documentação em separado pode ser utilizada apenas quando o programador necessitar de mais detalhes.

Algumas outras idéias interessantes que podemos utilizar são: incluir no comentário referências para outras funções (ou estruturas, variáveis globais, ...) relacionadas à função atual, e incluir referências bibliográficas. Desse modo acrescentaríamos ao comentário da função `insRN` as seguintes linhas:

```
// Veja também: rota, Nó.
// Referências: J. Szwarcfiter e L. Markenson, Estruturas de dados
// e seus algoritmos, Ed. LTC, pg. 156-160
```

Nessa função em particular é importante explicar como ela deve ser chamada externamente, então acrescentaríamos:

```
/* A chamada externa deve ser da forma insRN(x,ptv,NULL,NULL,1), onde
ptv aponta para a raiz de uma árvore rubro-negra de números inteiros. */
```

O que foi dito até agora também se aplica para comentários de classes. Um bom comentário de classe deve conter:

- o que a classe representa (que abstração ela representa).
- o significado das variáveis da classe (novamente para que não seja necessário descobrir os seus significados lendo o código).
- comentários individuais para cada método da classe (o que já foi discutido até aqui para métodos também se aplica aos métodos de uma classe).

Vamos ver um bom exemplo (em C++):

```
/* classe Stack: implementa uma pilha de números inteiros em um vetor
de 100 posições chamado data_buffer. A variável stack_ptr guarda
o índice do elemento localizado no topo da pilha, e vale -1 caso
a pilha esteja vazia.
- Referências: S. Holzner, Programando em C++, Ed. Campus,
  pg. 89-98 */

class Stack {

    int data_buffer[100];
    int stack_ptr;

    public:
      void init ();
      int pop ( int * pop_to );
      int push ( int push_this );

} my_stack;

// init: inicialização da pilha.
void Stack::init() {
```

```
    stack_ptr = -1;
}

/* push: recebe um inteiro push_this e o insere no topo da pilha.
Devolve 1 se a operação foi concluída com sucesso, 0 c.c. (caso
a pilha esteja cheia). */

int Stack::push ( int push_this ) {
    if (stack_ptr >= 99) return 0;
    else {
       data_buffer[++stack_ptr] = push_this;
       return 1;
    }
}

/* pop: remove o valor do topo da pilha. O valor removido é salvo
no endereço apontado pelo parâmetro pop_to. Devolve 1 se o valor
foi removido com sucesso, 0 c.c. (a pilha está vazia). */

int Stack::pop ( int * pop_to ) {
    if (stack_ptr == -1) return 0;
    else {
       *pop_to = data_buffer[stack_ptr--];
        return 1;
    }
}
```

Note que a função `init` é tão curta e que o que ela faz é tão óbvio que praticamente ela não foi comentada. Mas funções desse tipo são exceções. Normalmente você precisa de comentários do mesmo calibre das funções `pop` e `push`.

Resta falar agora sobre os comentários que ficam ao longo do código. Do mesmo modo que comentários de classes e funções são pequenos manuais das classes e das funções, estes comentários devem ser pequenos manuais das partes relevantes do código, e devem favorecer a compreensão rápida das linhas de código em questão. O programador deve usar o seu bom senso para não comentar o que é óbvio demais e para não dificultar a leitura do código devido ao excesso de comentários.

Usando a função `push` como exemplo, poderíamos acrescentar a ela as seguintes linhas de código:

```
int Stack::push ( int push_this ) {
    if (stack_ptr >= 99) // A pilha está cheia --
       return 0;         // retorne erro.
    else {               // Senão guarde o dado.
       data_buffer[++stack_ptr] = push_this;
       return 1;
    }
}
```

Na verdade a função `push` é tão curta que é redundante acrescentar esses comentários dado o comentário que a precede. Mas isso é só um exemplo.

Do mesmo modo que é importante explicar o significado dos parâmetros das funções e das variáveis das classes, pode ser importante explicar o significado das variáveis locais do código. Compare, por exemplo, os dois trechos de código, um sem e outro com comentários:

```
int main () {

  int i, j, k, tmp, ** IMAGEM, ** maux, M, N;


        . . .

}
```

```
int main () {

  int i, j, k, tmp,  // variaveis auxiliares.
    ** IMAGEM,       // matriz que guardará a imagem de entrada.
    ** maux;         // matriz auxiliar.
    M, N;            // dimensao da matriz IMAGEM e maux
                     // (N linhas, M colunas).

        . . .

}
```

3 Resumo

Em resumo, para cada método escreva:

- o que o método faz.
- o que o método recebe (i.e., os significados dos seus parâmetros).
- o que o método devolve (i.e., os significados dos valores de retorno, se existirem).

Para cada classe, escreva:

- o que a classe representa (que abstração ela representa).
- o significado das variáveis da classe.
- comentários individuais para cada método da classe.

Tanto os comentários de métodos como os de classes podem conter, opcionalmente, detalhes de implementação, referências bibliográficas e referências para outras partes do código.

No corpo do código, comente as partes relevantes. Cuidado para não poluir demais o código.

4 Consequências

Como consequências da adoção das idéias sugeridas para criação de comentários podemos ter:

- diminuição no tempo de aprendizado de uso do código.
- diminuição no tempo de entendimento da implementação do código (ou de parte da implementação, dependendo do nível de detalhe do comentário).

Algumas desvantagens:

- o tamanho do comentário pode ficar maior que o tamanho do código, caso o código seja pequeno.
- alterar o código me obriga a alterar o seu comentário.

5 Referências

Este padrão foi inspirado na página **Documentação** do professor Paulo Feofillof :

`http://www.ime.usp.br/˜pf/algoritmos/aulas/docu.htm`