CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
PROJETO E ANÁLISE DE ALGORITMOS

Última alteração: March 9, 2006

Solução e Documentação de Trabalhos Práticos

Apresentação

A apresentação do trabalho é importante. Hoje, o usuário ou cliente do seu trabalho é o professor ou o monitor do curso. Amanhã será seu chefe ou um cliente para o qual você estará desenvolvendo um produto.

Coloque toda a documentação num só documento, utilizando papel do mesmo tipo, de preferência tamanho A4, grampeado. Nunca rasure a listagem do programa ou os resultados de testes. Desenhos das estruturas de dados, títulos, comentários adicionados à documentação, etç podem ser feitos a mão. No entanto, é recomendável que você utilize um editor e/ou formatador de textos.

Documentação

A documentação do programa é como um pequeno artigo que explica o que o programa faz, como faz, e apresenta conclusões obtidas sobre o trabalho. A documentação é um documento à parte e não deve ser escrita no programa fonte.

A documentação a ser entregue deve conter pelo menos:

Descrição sucinta sobre o desenvolvimento do trabalho.
Uma explicação sobre as decisões de implementação tomadas, uma visão geral do funcionamento do programa, comentários sobre os testes executados, etc.

Descrição dos módulos e sua inter-dependência.
Uma breve descrição de cada módulo bem como um diagrama, por exemplo, mostrando a relação de dependência entre eles. Note que esta parte certamente estará relacionada com os TADs.

Descrição dos TADs e as estruturas de dados utilizadas.
Uma explicação sobre os TADs definidos, as operações disponíveis e como os TADs são implementados. Você pode fazer essa descrição utilizando desenhos ou escrevendo.

Descrição do formato de entrada dos dados.
Uma descrição simples e clara dizendo quais são os dados de entrada e como o programa irá recebê-los. Por exemplo:

> "A entrada para o programa consiste de um conjunto de descrição dos edifícios. Em cada linha haverá somente uma descrição. Cada descrição é composta por três números inteiros separados por um ou mais brancos na seguinte ordem: coordenada esquerda do edifício, altura do edifício, coordenada direita do edifício."

Descrição do formato de saída dos dados.
Uma descrição simples e clara dizendo como o programa apresentará os resultados ao usuário. Por exemplo:

> "O programa irá gerar uma sequência de números representado a linha do horizonte. Números que estiverem nas posições ímpares representam coordenadas e números que estiverem nas posições pares alturas."

Explicação sobre como utilizar o programa.

Por exemplo:

> "Para executar o programa da linha do horizonte, digite na linha de comando: lh.exe arqIn arqOut arqErro, onde arqIn é o arquivo que contém os dados de entrada, ..."

Estudo da complexidade do programa.

O estudo da complexidade deve analisar os principais procedimentos e/ou funções do programa mostrando qual é a sua complexidade final. Não se esqueça de especificar o que significa o parâmetro "n" que aparece nos estudos de complexidade. Por exemplo:

> "Seja n o número de cidades. O loop no procedimento GeraLista é execuatdo n vezes. Este loop só contém comandos de leitura e atribuição que são O(1). Logo, a complexidade deste procedimento é O(n) no pior caso."

Listagem do programa fonte.

Listagem dos testes executados.

A listagem dos testes deve conter os dados recebidos pelo programa (dados de entrada) e os resultados apresentados (dados de saída).

Programa fonte

Procure observar os seguintes aspectos no seu programa fonte:

Modularidade.

Não faça programas de um módulo só. Divida as tarefas a serem executadas em módulos. Utilize TADs.

Comentários.

Escreva os comentários no momento que estiver escrevendo o algoritmo. Um programa mal documentado é um dos piores erros que um programador pode cometer, e o sinal de um amador (mesmo com 10 anos de experiência). O melhor momento para se escrever os comentários é aquele em que o programador tem maior intimidade com o algoritmo, ou seja, durante a sua confecção.

> Os comentários devem acrescentar alguma coisa de útil, não apenas frasear o código.

O código nos diz como um certo problema está sendo resolvido.

Os comentários deverão dizer o que está sendo feito em cada passo. Use comentários no prólogo. É muito importante a colocação de comentários no prólogo de um programa ou de cada módulo, para explicar o que ele faz e fornecer instruções para seu uso. Poderiam ser colocados comentários dos seguintes tipos:

- O que faz o programa ou módulo;
- Como chamá-lo ou utilizá-lo;
- Significado dos parâmetros, variáveis de entrada, de saída e variáveis mais importantes;
- Arquivos utilizados;
- Outros módulos utilizados;
- Métodos especiais utilizados, com referências nas quais possa se encontrar mais informações;
- Avaliação do tempo de processamento e memória requeridos;
- Autor, data de escrita e última atualização;
- etc.

Identação. Utilize identação para mostrar a estrutura lógica do programa.

A identacao nao deve ser feita de forma caotica. Crie algumas regras básicas de identação e procure seguí-las ao escrever um programa.

Passagem de parâmetros.

Procure ser consistente na ordem e no tipo de passagem de parâmetros. Por exemplo:

"Suponha que três procedimentos diferentes têm que acessar os dados de uma mesma tabela. Não faz sentido se um dos procedimentos receber a tabela por valor, o outro por referência e o último acessar a tabela como variável global."

Variáveis globais.
Evite ao máximo a utilização de variáveis globais, por que elas compartilham dados entre as diversas partes do programa de uma maneira que não é explícita, o que pode levar a erros difíceis de serem achados.

Nomes de variáveis.
Escolha nomes representativos. Os nomes de constantes, tipos, variáveis, procedimentos, funções, etc. devem identificar o melhor possível o que representam. Por exemplo, X := Y + Z é muito menos claro que Preco := Custo + Lucro.

Utilize espacos em branco para melhorar a legibilidade.
Espaços em branco são valiosos para melhorar a aparência de um programa. Por exemplo:

- Deixar uma linha em branco entre as declarações e o corpo do programa;
- Deixar uma linha em branco antes e depois dos comentários;
- Separar grupos de comandos que executam funções lógicas distintas por uma ou mais linhas em branco;
- Utilizar brancos para indicar precedência de operadores, ao invés de A+B * C e bem mais legível a forma A + B*C;
- etc.

Um comando por linha é suficiente. A utilização de vários comandos por linha e prejudicial por vários motivos, dentre eles destacam-se o fato do programa tornar-se mais ilegível e ficar mais difícil de ser depurado.

Utilize parêntesis para aumentar a legibilidade e prevenir-se contra erros.

Testes

Procure fazer testes relevantes como, por exemplo, aqueles que verificam casos extremos e casos de exceções.

Desenvolvimento do programa

Não comece a desenvolver o programa antes que o problema esteja bem definido.
Uma das partes mais importantes da documentação e a especificação do problema usando alguma notação: escrita, gráfica, etc. O ato de escrever a especificação do problema e muito importante: ajuda na sua total compreensão e evita o esquecimento de detalhes relevantes. E preferível realismo no início do que ter que efetuar mudancas quando o programa atinge a "puberdade".

Após começar o desenvolvimento do programa procure nao mudar a especificação do problema.
A única razão para mudar a especificação de um problema após comecar o desenvolvimento deste deve ser para corrigir algum erro de especificação. Se a especificação for mudada constantemente o programa nunca ficará pronto além de acarretar mais custos.

Obviamente, isto não significa que não se deve alterar um programa depois que ele estiver pronto. É muito importante ter em mente que a maior parte dos programas desenvolvidos sofrerá algum tipo de modificação no futuro. Por essa razão, programas devem ser desenvolvidos de tal forma a minimizar o impacto introduzido pelas modificações.

A codificação deve ser feita de forma tão simples quanto possível.
Código complexo, sofisticado ou não usual atrapalha a legibilidade, depuração e modificação.

Estabeleça objetivos realistas o mais cedo possível.

Todo projeto de programação usualmente tem alguns objetivos que devem ser estabelecidos e colocados no papel. Tais objetivos variam naturalmente de projeto para projeto. Alguns itens a serem considerados são:

- Data de término;
- Facilidade de uso;
- Nível de confiabilidade (difícil de ser estimado);
- Limites de memória e tempo de execução;
- Generalidade;
- etc.

Quando se trabalha em equipe, o estabelecimento explícito dos objetivos irá garantir que todos trabalhem para se alcançar os mesmos objetivos.

Desenvolva cuidadosamente o programa usando alguma técnica de projeto como refinamento sucessivo.

Escrever diretamente qualquer código pode produzir uma barreira psicológica que poderá inibir futuros melhoramentos. Use alguma técnica de desenvolvimento de programas como a técnica de refinamento sucessivo.

Erros encontrados durante o desenvolvimento são relativamente fáceis de serem corrigidos comparados com erros encontrados durante os testes.

Um programa modesto que funciona é mais útil que um super-ambicioso que nunca funciona.

Muitas vezes é tentador fazer um programa com várias opções e melhorias. Mas nada adianta se esse programa nunca fica pronto para ser executado. Lembre-se que você não vai ter todo o tempo do mundo paraescrever seu programa. Por essa razão seja realista ao decidir o que fazer. Uma boa estratégia é ter uma versão simples que funciona e faz o que foi pedido e se houver tempo, introduzir novas opções e melhorias.

Selecione os algoritmos cuidadosamente.

Existem algumas regras que você deve procurar seguir ao fazer um programa:

- Não reprograme um programa, procedimento ou função que está pronto. A solução não é utilizar o primeiro algoritmo disponível. É muito importante que se entenda o módulo escolhido, ou seja, como funciona o algoritmo utilizado, quantidade de tempo e espaço necessários à execução desse módulo, etc.

  O que está sendo dito é que ninguém inventa todos os algoritmos que precisa utilizar em Ciência da Computação. O importante é saber e entender muito bem os algoritmos necessários à solução do seu problema.

- Não codifique o primeiro algoritmo que lhe vier à mente. O melhor algoritmo a ser utilizado depende dentre outros fatores da aplicação onde será utilizado. Por essa razão procure conhecer os possíveis algoritmos que podem ser usados na solução do problema. Só assim você terá certeza que escolheu um bom algoritmo ou não.

Escolha a representação dos dados adequada ao problema.

Uma boa apresentação dos dados pode eliminar páginas de código. A representação dos dados deve ser escolhida com base nas principais operações que serão feitas.

Utilize uma linguagem de programação adequada.

Se a linguagem de programação não é adequada ao problema a ser resolvido, certamente surgirão problemas na programação, na eficiência e na depuração do programa.

Evite que o programa seja dependente de dados particulares.

Um programa dependente de dados particulares, além de ser mais restrito, dificulta mudanças. Um exemplo típico de dependências de dados é na manipulação de vetores. Por exemplo:

```
for i := 1 to 25 do
   readln(entrada, A[i]);
Soma := 0;
for i := 1 to 25 do
   Soma := Soma + A[i];
```

O trecho de programa acima só funciona para um vetor com 25 elementos. No caso de se desejar manipular um número diferente de elementos, deve-se percorrer o programa modificando cada 25, o que é inconveniente. Além do mais pode-se saltar uma das constantes a ser modificada; ou mesmo, modificar uma constante com o mesmo valor, a qual não se deveria modificar. Uma solução melhor seria:

```
const N = 25;
...
for i := 1 to N do
   readln(entrada, A[i]);
Soma := 0;
for i := 1 to N do
   Soma := Soma + A[i];
```

Procure utilizar constantes, definir tipos, utilizar tipos abstratos de dados para evitar um problema similar ao apresentado acima.

Os formatos de entrada e saída devem ser bem legíveis.

Quando se projeta os formatos de entrada e saída deve-se ter sempre em mente o usuário. É importante definir uma ordem de variáveis e formatos de dados para entrada que sejam os mais naturais para o usuário. Isto evitará erros e facilitará o usos do programa.

A saída deve ser bem legível, sem que haja necessidade de se consultar o código do programa.

A facilidade de uso (entrada) e relatórios atraentes (saída) serão os itens que, em última instância, determinarão o julgamento do usuário quanto à capacidade de programador.

Se o programa não funciona, não interessa sua eficiência.

Um programa super eficiente, mas não confiável, dificilmente pode ser convertido em um programa confiável. Mas um programa confiável e bem estruturado, mesmo que ineficiente, pode ser otimizado.

A legibilidade é, usualmente, mais importante que a eficiência. A legibilidade facilita o entendimento, permite possíveis modificações, inclusives a introdução de eficiência.

Antes de otimizar procure saber quão eficiente o programa precisa ser.

A eficiência depende da aplicação, sendo mais relevante em alguns casos do que em outros. Preferencialmente, o nível de eficiência que se quer de um programa deve constar na sua especificação.

Em geral, programas utilizados com muita frequência devem ser mais eficientes do que os utilizados raramente.

About this document ...

The command line arguments were:
latex2html `-split 0 roteiro`

The translation was initiated by Nivio Ziviani on 2006-03-09

Nivio Ziviani 2006-03-09