

Como Abordar um Problema de Programação

Vinícius José Fortuna

(aluno de mestrado do IC-UNICAMP, participante da IOI99)

Fonte: http://olimpiada.ic.unicamp.br/programacao/programacao_junior/dicas

Projete seu programa antes de programá-lo

Nunca comece a programar a partir do nada. Deve-se sempre esquematizar alguns pseudo-códigos explicando o que o seu programa vai fazer (em um nível mais elevado) antes de começar a programar. A única exceção é quando se trata de um código que você já escreveu diversas vezes (p/ ex.: encontrar um elemento em um vetor).

Quando se começa a escrever um programa sem ter pensado nele antes, fica difícil visualizá-lo como um todo. Criando um rascunho prévio do programa, podem aparecer várias abordagens do problema e as dificuldades ficam mais fáceis de serem superadas. Esquematizar o programa ajudar a fixar exatamente o que se deseja e economiza-se bastante tempo em frente ao monitor na tentativa de escrever um programa que cumpra o desejado.

Escreva um código legível

Escrever um código legível é muito importante para facilitar o entendimento de um programa. Até para o próprio criador do código. Em programa claro e auto-explicativo fica mais difícil se perder e torna muito mais fácil a depuração.

Comente seu código enquanto escreve, não depois

Comentários são ferramentas muito úteis para tornar o código mais legível. É interessante comentar tudo que não seja muito claro. Não comente algo que seja óbvio (p/ ex.: "i := 0 { Atribui o valor 0 à variável i }"). Comente algo como: "x:= 40 - Length(frase)/2 { x recebe a posição para frase ficar centralizada }".

Em programa muito grandes ou complicados, é interessante criar um cabeçalho comentado em cada função, definindo exatamente o que espera-se que ela faça, quais suas entradas e quais suas saídas. O pseudo-código rascunhado pode ser muito útil para isso. Agindo assim, não se precisa ler diversas linhas de código para saber o que uma função faz.

É recomendável que se escreva os comentários enquanto se escreve o programa, pois é menos provável que se escreva alguma coisa útil ou significativa depois. Escreva enquanto programa e seus comentários serão muito mais completos.

Utilize margens e indentação apropriadamente

A cada novo *loop*, expressões condicionais, definição de funções e blocos de comandos, seu código deve ser indentado um nível mais à direita (pressione [TAB] ou a barra de espaço algumas vezes). Esteja certo de voltar ao nível de indentação anterior quando terminar o bloco.

Linhas em branco também são muito úteis para aumentar a legibilidade do seu código. Uma ou duas linhas entre as definições de funções e procedimentos e uma linha entre a definição de variáveis e o código irão separar claramente cada parte, o que torna a identificação delas mais rápida. Isso torna o

código bem mais claro.

Use nomes sugestivos para variáveis, funções e procedimentos

O código fica incrivelmente mais difícil de ser depurado quando variáveis importantes se chamam p, t, mal, qq, e assim por diante. Deve-se sempre utilizar nomes sugestivos para as variáveis, funções e procedimentos. O nome deve dar idéia do que a variável representa ou o que a função ou procedimento fazem. Por exemplo, se você quer armazenar o número de alunos em uma variável, pode-se usar num_alunos. Se for uma função que calcula o salário médio, pode-se nomeá-la calc_SalarioMedio().

Utilize funções e procedimentos curtos e objetivos

Evite sempre funções/procedimentos grandes que englobem todo tipo de processamento. Separe algoritmos distintos em suas próprias funções/procedimentos. Projete sua grande função/procedimento em várias pequenas, de forma que seu programa fique mais fácil de ler e entender.

Dessa forma, cada parte do seu programa fica bem definida e torna-se muito mais fácil escrevê-lo, pois pode-se fazê-lo passo a passo. Dessa forma, a cada parte que se termina, pode-se verificar se ela está correta. Além disso a localização de um problema no programa também fica facilitada, pois ele se restringirá a um bloco menor de código.

Conclusão:

Lembre-se que a maior parte do tempo que se gasta programando é corrigindo e modificando código existente. Relativamente pouco tempo é realmente utilizado para adicionar coisas novas. Isso significa que você gastará muito tempo lendo o seu código, então faz sentido gastar algum tempo aprendendo a escrever um código legível. Código legível é fácil de escrever, fácil de depurar e fácil de manter. Você realmente sai ganhando!

Se estiver confuso na hora da depuração

Se você estiver confuso ao tentar procurar algum problema no seu programa, tente explicá-lo para você mesmo. Dessa forma é possível notar inconsistências ou fugas ao algoritmo planejado.

Caso isso não resolva, pode-se tentar executar o programa no papel. Isso se aplica tanto a códigos que você escreveu e não está mais entendendo como a códigos pegos de outros. Funciona da seguinte maneira: Pegue uma folha em branco e liste todas as variáveis usadas no programa. Siga linha por linha do código, escrevendo o valor das variáveis enquanto elas mudam, como se você fosse o computador. Pode-se usar uma calculadora para ajudar nas contas. Anote todas as saídas em uma folha à parte. Após algumas poucas iterações a estrutura básica do algoritmo e sua intenção devem ficar claras. Tome cuidado, pois nem sempre o código funciona do jeito que nós pensamos que funciona.

Guia prático para resolução de problemas de programação

1) Entender o problema

- Esteja certo de que tenha entendido o problema;

- O que é a entrada?
- O que é a saída?

2) Resolver o problema à mão

- Resolva pequenas instâncias do problema à mão;
- O que acontece?
- Pense em casos variados;
- Pense em como (qual algoritmo) você utilizou para resolver o problema.

3) Definir o algoritmo

- Defina precisamente o algoritmo a ser utilizado
- Rascunhe as etapas do programa

4) Programar

- Como escrever o algoritmo na linguagem utilizada?
- Que estrutura de dado utilizar?¹
- Divida o programa em partes menores (modularizar);
- Escreva um programa de fácil leitura;
- Pense nos casos patológicos.²

4) Depurar

- Explique o programa para si mesmo;
- Por que funciona?
- A leitura de dados está sendo feita corretamente?
- Variáveis inicializadas?
- Verificar casos patológicos;
- Localizar o erro restringindo os blocos de códigos (cercando o erro)
- Comandos e *loops* aninhados corretamente?

Observações:

1) Que estrutura utilizar?

Qual a melhor forma de representar as variáveis do problema. Variáveis simples? Vetores? Matrizes? Registros? Alguns vetores? Vetores de registro? Registros de vetores? São muitas as estruturas utilizáveis. Deve-se escolher uma que seja conveniente e que não venha trazer complicações mais adiante.

2) Pense nos casos patológicos

Os casos patológicos ocorrem quando a propriedade que seu programa utiliza não vale para alguns valores. Normalmente são o zero, um, valores iniciais ou finais. Por exemplo, em uma função que calcula a potência de um número n pelo expoente e . Para isso pode-se multiplicar o número n e vezes. Nesse caso pode-se ter problemas quando o valor de e for zero, caso que deve ser tratado especialmente (considerando a resposta padrão como 1, por exemplo). Para ilustrar melhor, imagine o caso em que deseja-se verificar se um vetor está ordenado em ordem não-decrescente. Para isso basta verificar se $v[n] \leq v[n+1]$ para todos os elementos, exceto o último, pois para ele essa propriedade não tem sentido. Os casos patológicos são causa de grande parte dos problemas, especialmente quando se trabalha com ponteiros.

Referências

- www.gamedev.net
- Skiena, Steven S. "The Algorithm Design Manual", Telos, 1997