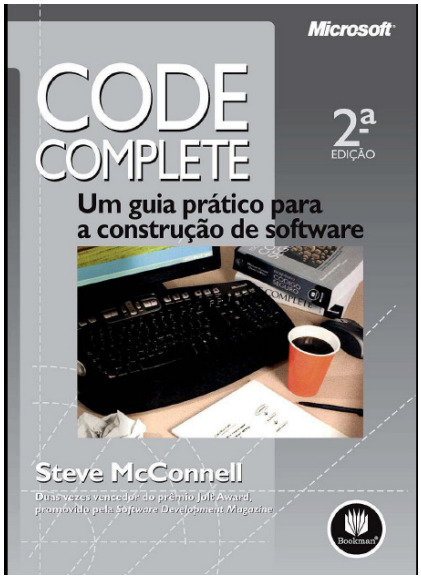
Construção de Software

Variáveis

Prof. Rubens de Castro Pereira, Me.
rubens@inf.ufg.br

INSTITUTO DE INFORMÁTICA
UFG



Code Complete
Um guia prático para a construção de software, 2ª ed., Steven McConnell
Bookman

Cap. 11 – O poder dos nomes de Variáveis, pág. 285 a 314

Variáveis
Considerações sobre bons nomes

Exemplo em Java de maus nomes de variável

```
x = x - xx;
xxx = fido + SalesTax( fido );
x = x + LateFee( x1, x ) + xxx;
x = x + Interest( x1, x );
```

- O quê está acontecendo com o código fonte acima?
- O que x, xx, xxx, x1 e fido significam?

Variáveis
Considerações sobre bons nomes

Exemplo em Java de maus nomes de variável

```
x = x - xx;
xxx = fido + SalesTax( fido );
x = x + LateFee( x1, x ) + xxx;
x = x + Interest( x1, x );
```

Exemplo em Java de bons nomes de variável

```
balance = balance - lastPayment;
monthlyTotal = newPurchases + SalesTax( newPurchases );
balance = balance + LateFee( customerID, balance ) + monthlyTotal;
balance = balance + Interest( customerID, balance );
```

Variáveis
Considerações sobre bons nomes

- O nome de uma variável deve descrever de forma total e precisa e entidade que representa.
- Usar palavras agrupadas para nomear variáveis.
- Nomes não podem ser ambíguos, de natureza obscura ou gerar dúvidas na sua leitura.
- Devem ser o mais específico possível.

Tabela 11-1 Exemplos de nomes de variável bons e ruins

Objetivo da variável	Bons nomes, bons descritores	Nomes ruins, descritores ruins
Total geral dos cheques gravados até a presente data	*totalGeral, totalDeCheques*	*gravados, tc, cheques, TTCHQ, x, x1, x2*
Velocidade de um trem-bala	*velocidade, velocidadeDoTrem, velocidadeEmKph*	*velt, v, tv, x, x1, x2, trem*
Data atual	*dataAtual, dataDeHoje*	*da, atual, a, x, x1, x2, data*
Linhas por página	*linhasPorPágina*	*lpp, linhas, l, x, x1, x2*

Variáveis
Considerações sobre bons nomes

- Um bom nome mnemônico geralmente:
  - representa o problema, não a sua solução.
  - Tende a expressar mais ***o que*** do que ***como***.
  - Exemplo:
    - Dados de registro de funcionário: registroEntrada ou dadosFuncionario?
    - Status da impressora: bitFlag ou impressoraPronta?

Variáveis
Considerações sobre bons nomes

- Nomes muitos curtos não possuem significado
- Nomes muito longos são de difícil digitação podendo prejudicar a estrutura visual do código fonte.
- O comprimento ideal dos nomes pode variar entre 8-20, 9-15 ou 10-16.

Tabela 11-2 Nomes de variável longos demais, curtos demais ou adequados

Longos demais:	*númeroDePessoasNaEquipeOlímpicaDosEUA*
	númeroDeLugaresNoEstádio
	númeroMáximoDePontosNasOlimpíadasModernas
Curtos demais:	*n, np, nme*
	n, nl, ndlne
	m, mp, max, pontos
Adequados:	*numMembrosNaEquipe, contMembrosDaEquipe*
	numLugaresNoEstádio, contagemDeLugares
	maxPontosDaEquipe, recordeDePontos

7

Variáveis
Considerações sobre bons nomes

- Qualificadores de variáveis com valores calculados como Max, Min, Media, Total, etc, podem ser colocados no final do nome da variável:
  - gastoTotal
  - despesaMedia
  - notaFinal
  - consumoMax
  - consumoMin
- A parte mais forte do nome fica em primeiro lugar e depois o modificador
- Auxilia na padronização e compreensão

Variáveis
Considerações sobre bons nomes

- Opostos comuns em nomes de variáveis:
  - inicio/fim
  - primeiro/ultimo
  - bloqueado/desbloqueado
  - min/max
  - proximo/anterior
  - antigo/novo
  - aberto/fechado
  - visivel/invisivel
  - fonte/alvo
  - origem/destino
  - cima/baixo
- Outros exemplos:
  - inicioReserva
  - fimReserva
  - valvulaAberta
  - valvulaFechada
  - proximoElemento
  - anteriorElemento
  - novoAluno
  - antigoAluno

Atribuindo nomes a tipos de dados específicos
Índice de loop

Exemplo em Java de um nome de variável de *loop* simples

```
for ( i = firstItem; i < lastItem; i++ ) {
   data[ i ] = 0;
}
```

Exemplo em Java de um bom nome descritivo de variável de *loop*

```
recordCount = 0;
while ( moreScores() ) {
   score[ recordCount ] = GetNextScore();
   recordCount++;
}

// linhas usando recordCount
...
```

Exemplo em Java de bons nomes de *loop* em um *loop* aninhado

```
for ( teamIndex = 0; teamIndex < teamCount; teamIndex++ ) {
   for ( eventIndex = 0; eventIndex < eventCount[ teamIndex ]; eventIndex++ ) {
      score[ teamIndex ][ eventIndex ] = 0;
   }
}
```

Atribuindo nomes a tipos de dados específicos
Variáveis de Status

Exemplos em C++ de *flags* de natureza obscura

```
if ( flag )...
if ( statusFlag & 0x0F )...
if ( printFlag == 16 )...
if ( computeFlag == 0 )...

flag = 0x1;
statusFlag = 0x80;
printFlag = 16;
computeFlag = 0;
```

CODIFICAÇÃO HORRÍVEL

Exemplos em C++ de melhor uso de variáveis de *status*

```
if ( dataReady )...
if ( characterType & PRINTABLE_CHAR )...
if ( reportType == ReportType_Annual )...
if ( recalcNeeded == True )...

dataReady = true;
characterType = CONTROL_CHARACTER;
reportType = ReportType_Annual;
recalcNeeded = false;
```


Atribuindo nomes a tipos de dados específicos
Variáveis de Status

Declarando variáveis de *status* em C++

```
// valores de CharacterType
const int LETTER = 0x01;
const int DIGIT = 0x02;
const int PUNCTUATION = 0x04;
const int LINE_DRAW = 0x08;
const int PRINTABLE_CHAR = ( LETTER | DIGIT | PUNCTUATION | LINE_DRAW );

const int CONTROL_CHARACTER = 0x80;

// valores de ReportType
enum ReportType {
   ReportType_Daily,
   ReportType_Monthly,
   ReportType_Quarterly,
   ReportType_Annual,
   ReportType_All
};
```

Atribuindo nomes a tipos de dados específicos
Variáveis Temporárias

Exemplo em C++ de um nome de variável "temporária" não-informativo

```
// Calcula as raízes de uma equação quadrática.
// Isso presume que (b^2-4*a*c) é positivo.
temp = sqrt( b^2 - 4*a*c );
root[0] = ( -b + temp ) / ( 2 * a );
root[1] = ( -b - temp ) / ( 2 * a );
```

Exemplo em C++ com um nome de variável "temporária" substituído por uma variável real

```
// Calcula as raízes de uma equação quadrática.
// Isso presume que (b^2-4*a*c) é positivo.
discriminant = sqrt( b^2 - 4*a*c );
root[0] = ( -b + discriminant ) / ( 2 * a );
root[1] = ( -b - discriminant ) / ( 2 * a );
```

O nome da variável é preciso e descritivo

Atribuindo nomes a tipos de dados específicos
Variáveis Booleanas

- Nomes booleanos típicos:
  feito – indica que algo está pronto
  erro – indica ocorrência de um erro
  encontrado – indica se um valor foi encontrado
  sucesso / **ok** – indica que a ação foi realizada com sucesso
- Nomes ruins, pois não indicam verdade ou falsidade:
  - status, arquivoDestino, gravacao
- Use nomes que indiquem verdadeiro ou falso:
  - statusOk, arquivoDestinoDisponivel, gravacaoSucesso
- Use nomes de variáveis booleanas positivas:
  - feito, encontrado, sucesso
  - O contrário é de difícil compreensão:

```
if (!notFound) {
    ...
}
```

Atribuindo nomes a tipos de dados específicos
Tipos Enumerados

- Para linguagens não OO, useo prefixo do nome do grupo para nomear os membros:
  - Cor_Vermelho, Cor_Preto
- Para linguagens OO é desnecessário o nome da classe como prefixo:
  - Cor.Vermelho, Cor.Preto, etc

Atribuindo nomes a tipos de dados específicos
Constantes

- Nomeie a entidade que a constante representa:
  - Errado: CINCO
  - Certo: NUMERO_TENTATIVAS

Variáveis
Convenções de atribuição de nomes

- Para quê?
  - Possibilitam decisões globais ao invés de locais, podendo concentrar nas características mais importantes do código.
  - Auxiliam a transferir conhecimentos entre projetos.
  - Auxiliam ao aprendizado do código mais rapidamente para novos projetos.
  - Reduz a proliferação de nomes diferentes para a mesma coisa (Ex: pontosTotais ou totalDePontos).
  - Compensam as limitações da linguagem diferenciando dados locais, de classe e globais, além de incorporar informações sobre o tipo do dado.
  - Auxiliam na associação a itens relacionados, principalmente para linguagens não OO.

Segredo: Qualquer convenção é melhor do que nenhuma!

Variáveis
Convenções de atribuição de nomes

- Quando?
  - Projeto com vários desenvolvedores.
  - Projeto necessita de alterações ou entra em manutenção.
  - Código sobre revisões.
  - Sistema com vida longa.
  - Existem vários termos incomuns no projeto e necessita padronizar a codificação.

Variáveis
Convenções Informais de atribuição de nomes

- Diretrizes para convenção independente da linguagem:
  - Diferencie entre nomes de variável/objeto e nomes de rotina (método, função ou procedimento):
    - nomeVariavel ou nomeObjeto
    - NomeMetodo ou NomeFuncao ou NomeProcedimento
  - Diferencie classe/tipo e objeto/variável com as seguintes opções:

Diretrizes para convenção independente da linguagem

Diferencie classe/tipo e objeto/variável com as seguintes opções:

Opção 1: diferenciar tipos e variáveis com a letra maiúscula inicial

```
Widget widget;
LongerWidget longerWidget;
```

Opção 2: diferenciar tipos e variáveis com todas as letras maiúsculas

```
WIDGET widget;
LONGERWIDGET longerWidget
```

Opção 3: diferenciar tipos e variáveis com o prefixo "t_" para tipos

```
t_Widget Widget;
t_LongerWidget LongerWidget;
```

Opção 4: diferenciar tipos e variáveis com o prefixo "a" para variáveis

```
Widget aWidget;
LongerWidget aLongerWidget;
```

Opção 5: diferenciar tipos e variáveis usando nomes mais específicos para as variáveis

```
Widget employeeWidget;
LongerWidget fullEmployeeWidget;
```

Diretrizes para convenção independente da linguagem

- Identifique:
  - Variáveis globais.
  - Variáveis membros (de classe).
  - Definições de tipos.
  - Constantes.
  - Definições de tipos enumerados (enum).
  - Parâmetros somente de entrada nas linguagens que não os impõem: parâmetros por referência.
  - Formate os nomes para melhorar a legibilidade (letras maiúsculas e minúsculas, sublinhados para separar palavras) → Cuidado em C, C++ e Java, pois maiúscula é diferente de minúscula.

Convenção de nomes para C

Tabela 11-4 Exemplos de convenções de atribuição de nomes para C

Entidade	Descrição
TypeName	As definições de tipo usam letras maiúsculas e minúsculas misturadas, com uma letra maiúscula no início.
GlobalRoutineName( )	As rotinas públicas são compostas de letras maiúsculas e minúsculas misturadas.
f_FileRoutineName( )	As rotinas que são privadas para um único módulo (arquivo) são prefixadas com *f_*.
LocalVariable	As variáveis locais são compostas de letras maiúsculas e minúsculas misturadas. O nome deve ser independente do tipo de dados subjacente e deve se referir ao que a variável representa.
RoutineParameter	Os parâmetros de rotina são formatados como as variáveis locais.
f_FileStaticVariable	As variáveis de módulo (arquivo) são prefixadas com *f_*.
G_GLOBAL_GlobalVariable	As variáveis globais são prefixadas com *G_* e um mnemônico do módulo (arquivo) que define a variável, com todas as letras maiúsculas – por exemplo, *SCREEN_Dimensions*.
LOCAL_CONSTANT	As constantes nomeadas que são privadas para uma única rotina ou módulo (arquivo) usam todas as letras maiúsculas – por exemplo, *ROWS_MAX*.
G_GLOBALCONSTANT	As constantes nomeadas globais usam todas as letras maiúsculas e são prefixadas com *G_* e um mnemônico do módulo (arquivo) que define a constante nomeada, com todas as letras maiúsculas – por exemplo, *G_SCREEN_ROWS_MAX*.
LOCALMACRO( )	As definições de macro que são privadas para uma única rotina ou módulo (arquivo) usam todas as letras maiúsculas.
G_GLOBAL_MACRO( )	As definições de macro globais usam todas as letras maiúsculas e são prefixadas com *G_* e um mnemônico do módulo (arquivo) que define a macro, com todas as letras maiúsculas – por exemplo, *G_SCREEN_LOCATION( )*.

Convenção de nomes para C++ e Java

Tabela 11-3 Exemplos de convenções de atribuição de nomes para C++ e Java

Entidade	Descrição
ClassName	Os nomes de classe são compostos de letras maiúsculas e minúsculas misturadas, com uma letra maiúscula no início.
TypeName	As definições de tipo, incluindo tipos enumerados e *typedefs*, usam letras maiúsculas e minúsculas misturadas, com uma letra maiúscula no início.
EnumeratedTypes	Além da regra anterior, os tipos enumerados são sempre expressos na forma plural.
localVariable	As variáveis locais são compostas de letras maiúsculas e minúsculas misturadas, com uma letra minúscula no início. O nome deve ser independente do tipo de dados subjacente e deve se referir ao que a variável representa.
routineParameter	Os parâmetros de rotina são formatados como as variáveis locais.
RoutineName( )	As rotinas são compostas de letras maiúsculas e minúsculas misturadas. (Bons nomes de rotina foram discutidos na seção 7.3.)
m_ClassVariable	As variáveis-membro que estão disponíveis para várias rotinas dentro de uma classe, e somente dentro de uma classe, são prefixadas com *m_*.
g_GlobalVariable	As variáveis globais são prefixadas com *g_*.
CONSTANT	As constantes têm todas as *LETRAS_MAIÚSCULAS*.
MACRO	As macros têm todas as *LETRAS_MAIÚSCULAS*.
Base_EnumeratedType	Os tipos enumerados são prefixados com um mnemônico de seu tipo de base, expresso no singular – por exemplo, *Color_Red*, *Color_Blue*.

Variáveis
Prefixos padronizados

Tabela 11-6 Exemplos de TDUs para um processador de textos

Abreviação do TDU	Significado
ch	Caractere (um caractere não no sentido da linguagem C++, mas no sentido do tipo de dados que um programa de processamento de textos usaria para representar um caractere em um documento)
doc	Documento
pa	Parágrafo
scr	Região da tela
sel	Seleção
wn	Janela

```
CH    chCursorPosition;
SCR   scrUserWorkspace;
DOC   docActive
PA    firstPaActiveDocument;
PA    lastPaActiveDocument;
WN    wnMain;
```

24

Variáveis
Prefixos padronizados - Semânticos

Tabela 11-7 Prefixos semânticos

Prefixo semântico	Significado
c	Contagem (como o número de registros, caracteres, etc.)
first	O primeiro elemento que precisa ser tratado em um *array*. O prefixo *first* é semelhante a *min*, mas relativo à operação corrente, em vez de ao *array* em si.
G	Variável global
I	Índice de um *array*
Last	O último elemento que precisa ser tratado em um *array*. O prefixo *last* é o oposto de *first*.
lim	O limite superior dos elementos que precisam ser tratados em um *array*. O prefixo *lim* não é um índice válido. Assim como *last*, *lim* é usado como o inverso de *first*. Ao contrário de *last*, *lim* representa um limite superior não inclusivo no *array*; *last* representa um elemento final válido. Geralmente, *lim* é igual a *last* + *1*.
m	Variável em nível de classe
max	O último elemento absoluto em um *array* ou outro tipo de lista. O prefixo *max* se refere ao *array* em si, em vez de a operações no *array*.
min	O primeiro elemento absoluto em um *array* ou outro tipo de lista.
p	Ponteiro

Variáveis
Nomes curtos legíveis

- Use abreviações-padrão que estão em uso comum
- Remova todas as vogais que não estejam no início:
  computador → cmptdr, tela → tl
- Remova os artigos entre as palavras: e, o, a
- Trunque firmemente após a primeira, segunda e terceira letra de cada palavra (o que for mais apropriado)
- Mantenha a primeira e última letras de cada palavra:
  cadastro → cdstro
- Use cada palavra significativa no nome, até um máximo de três palavras
- Remova sufixos inúteis como *indo, ando, ado, etc*
- Mantenha o som mais perceptível em cada sílaba
- Certifique-se de não mudar o significado da variável
- Itere por essas técnicas até abreviar o nome da variável para algo entre 8 a 20 caracteres ou o limite estabelecido pela linguagem de programação

Variáveis
Tipos de nomes a serem evitados

- Evite nomes ou abreviações que induzam ao erros: FALSO abreviação de "**F**eira de **A**mostra e **L**ançamento de **So**ftware"
- Evite nomes com significados semelhantes
- Evite variáveis com nomes semelhantes, mas significados diferentes
- Evite nomes que tenham sons semelhantes como *wrap* e *rap*
- Evite numerais nos nomes: tabela1, tabela2, nome1, endereco1
- Evite palavras grafadas erroneamente nos nomes
- Evite palavras que normalmente são grafadas de forma errada em inglês
- Não diferencie nomes de variáveis unicamente pelas letras maiúsculas e minúsculas
- Evite múltiplos idiomas
- Não use nomes que são totalmente desassociados ao que as variáveis representam
- Evite nomes contendo caracteres difíceis de ler: *hard2Read, 6RANDTOTAL*

Variáveis

Leitura extra aula:

1. Code Complete – Um guia prático para a construção de software, 2ª ed., Steven McConnell, Bookman

Parte III – Variáveis:

- Cap. 11 – O poder dos nomes de Variáveis, pág. 285 a 314