Guia de Nomenclatura Usada em Identificadores

As convenções empregadas na escrita de identificadores que integram as implementações em linguagem C deste livro serão descritas neste apêndice, que é dividido em duas seções. A primeira delas dedica-se a explicar brevemente as regras básicas seguidas na escrita das diversas categorias de identificadores. Por sua vez, a segunda seção deste apêndice explica o significado das terminações desses identificadores.

C.1 Regras Básicas de Escrita de Identificadores

Todos os identificadores usam como base a famosa **notação camelo** e seguem os princípios descritos nestas seção. Identificadores que representem categorias diferentes de componentes de um programa seguem notações diferentes de composição para facilitar a identificação visual de cada componente.

C.1.1 Variáveis e Parâmetros Formais

Variáveis e parâmetros formais usam as seguintes convenções na composição de seus nomes:

- O nome de uma variável (ou parâmetro) sempre começa com letra minúscula
- Se o nome de uma variável (ou parâmetro) for composto, utiliza-se letra maiúscula no início de cada palavra seguinte, incluindo palavra de ligação (p. ex., preposição e conjunção)
- Não são usados subtraços (i.e., _)
- Frequentemente, variáveis ou parâmetros que começam com *p* (p. ex., `pNovoNo`) são ponteiros, enquanto aquelas que começam com *i* (p. ex., `iChave`) representam índices

C.1.2 Macros e Constantes de Enumeração

Com respeito a macros (incluindo constantes simbólicas) e constantes de enumeração, adota-se a seguinte política de escrita de identificadores:

- Utilizam-se apenas letras maiúsculas
- Se um identificador for composto, utilizam-se subtraços para separar as palavras que o constituem (p. ex., `TAM_MAIOR_PALAVRA`).

C.1.3 Funções

As normas usadas na criação de nomes de funções são as seguintes:

- O nome de uma função reflete aquilo que a função faz ou produz. Uma função que retorna um valor é denominada pelo *nome* do valor retornado. Por exemplo, uma função que calcula o fatorial de um número é denominada apenas como *Fatorial* (e não, por exemplo, *CalculaFatorial* ou, pior, *RetornaFatorial*). Por outro lado, o nome de uma função que não retorna nada (i.e., cujo tipo de retorno é **void**) indica o tipo de processamento que ela efetua. Por exemplo, uma função que ordena uma lista de nomes é denominada de forma sucinta como *OrdenaNomes* ou, se for necessário ser mais prolixo, *OrdenaListaDeNomes*. Também, funções que retornam um dentre dois valores possíveis (p. ex., sim/não ou verdadeiro/falso) são nomeadas começando com *Eh* [representando *é* — p. ex., `EhLegal()`] ou *Sao* [representando são — p. ex., `SaoIguaisListasG()`].
- Cada palavra constituinte do nome de uma função começa por letra maiúscula e é seguida por letras minúsculas. Procedendo assim, não é necessário usar subtraço para separar as partes constituintes de um identificador.
- Funções que implementam algoritmos com nomes consagrados, mesmo que em inglês, adotam os nomes desses algoritmos. Por exemplo, a função que implementa o algoritmo **BubbleSort** é `BubbleSort()`.

C.1.4 Tipos

A notação adotada neste livro para identificadores de tipos dita que eles comecem com a letra *t*. Em seguida, procede-se como na criação de um nome de função (p. ex., `tAluno`).

C.1.5 Rótulos de Estruturas

A diferença entre a notação usada para tipos e aquela usada para rótulos de estruturas é que esses rótulos começam com *rot* (p. ex., `rotListaIdxD`, `rotNoArvoreBin`) em vez de *t*.

C.2 Glossário de Sufixos de Identificadores

Grande parte dos identificadores que aparecem em implementações deste livro tem uma **terminação** (**sufixo**) que indica a estrutura de dados ou operação à qual ele se refere. Por exemplo, identificadores que terminam em *Idx* (como `CriaPilhaIdx`) referem-se a listas indexadas. O leitor pode consultar a **Tabela C–1** para esclarecer alguma dúvida referente essas terminações. Sufixos que não aparecem nessa tabela devem ter significados óbvios no contexto em que eles se encontram. Por exemplo, parece evidente que o tipo `tNoPoli` refere-se a polinômios na conjuntura em que ele se encontra.

Capítulos	Terminação	Refere-se a...	Exemplos
7 e 8	`Idx`	Estrutura de dados indexada implementada usando array estático	❒ **Tipo:** `tListaIdx` ❒ **Função:** `ComprimentoListaIdx()`
9	❒ `IdxD` ❒ `IdxD2`	Estrutura de dados indexada implementada usando array dinâmico	❒ **Rótulo:** `rotListaIdxD` ❒ **Função:** `InsereEmOrdemIdxD2()`
10	`SE`	Estrutura de dados implementada usando lista simplesmente encadeada	❒ **Rótulo:** `rotNoLSE` ❒ **Função:** `EstaVaziaListaSE()` ❒ **Tipo:** `tNoPilhaSE`
10	`DE`	Estrutura de dados implementada usando lista duplamente encadeada	❒ **Rótulo:** `rotNoLDE` ❒ **Função:** `RemoveListaDE()`
10	`SEC`	Estrutura de dados implementada usando lista simplesmente encadeada circular	❒ **Tipo:** `tNoListaSEC` ❒ **Função:** `RemoveListaSEC()`
10	`DEC`	Estrutura de dados implementada usando lista duplamente encadeada circular	❒ **Rótulo:** `rotNoLDEC` ❒ **Função:** `EstaVaziaListaDEC()`
10	`DECCab`	Estrutura de dados implementada usando lista duplamente encadeada circular com cabeça	❒ **Tipo:** `tNoLDECCab` ❒ **Função:** `RemoveListaDECCab()`
11	❒ `G` ❒ `Gen`	Estrutura de dados ou função genérica	❒ **Tipo:** `tNoListaG` ❒ **Função:** `BubbleSortGen()`
12	`Bin`	Árvore binária	❒ **Rótulo:** `rotNoArvoreBin` ❒ **Função:** `ConstroiNoArvoreBin()`
12	`Cost`	Árvore binária costurada	❒ **Tipo:** `tArvoreCost` ❒ **Função:** `FilhoDireitoCost()`
12	`Huff`	Identificador usado na codificação de Huffman	❒ **Tipo:** `tNoArvoreHuff` ❒ **Função:** `ExibeCodigosHuff()`
10 e 12	`Ord`	Lista ordenada	❒ **Função:** `CriaListaSEOrd()`
4, 6 e 7	`Rec`	Função recursiva	❒ `BuscaBinariaRec()`

Tabela C–1: Sufixos de Identificadores