ソフトウェア工学 58－15
（１９８８．２．５）

プログラム変数の命名法

佐藤　匡正
ＮＴＴ㈱情報通信処理研究所

プログラムコードにおける変数名はプログラムを解読する上で重要な手がかりとなる。分かりやすい名前がつけられていれば保守はしやすい。変数にどの様な名前をつけるかはプログラマー個人に任せられている。プログラマー各人は、暗黙的ではあるがそれぞれ独自な命名法をもっている筈である。このような命名法を探り出せればプログラミングや保守に役立つ。

ここでは、この考えから実際のプログラムにおける変数名について名前を構成している名詞の略し方に着目した分析を行いプログラマーが暗黙に用いている命名法を推定した。この結果、①合成語と単一語の略し方、②略し方のバリエーション、などを定量的に捉えることができた。

"Program Variables Naming Conventions" (in Japanese)

by Tadamasa SATOU
(Communications and Information Processing Laboratories, NTT, Yokosuka-shi Kanagawa 238-03 Japan)

Variable names in program source codes provide for important hints in reading programs. Reasonable names promote maintenance works very much. Naming conventions should be established. This paper presents a quantitative analysis of program variable names in commercial use programs. Through the analysis the construction and abbreviation of nouns that are included in a variable name are focused. As the result the following data are gotten;
i) naming conventions for complex words and single words
ii) variations in naming

1.序論

変数の名前をどのようにつけるかはプログラミングにおいては重要な事項のひとつである。変数の名前はプログラムコードにおいては主役ともいえる存在である。分かりやすい名前がつけられたプログラムコードはデバッグや保守がしやすい。変数の名前は管理するという立場と処理に意味を与えるという二通りの立場がある。管理する立場ではシステムで使っている名前が唯一であるようにする。処理に意味を与える立場では名前に変数のもつ役割や値の意味を盛り込む。変数にこのような名前が付いていれば難解なソースコードの解読の有力な手がかりとなるので読み易くなる。プログラマーは後者の処理に意味を与える立場から名前を付けようとする。ところが分かりやすい名前を付けるには苦労する。名前を付けるための具体的な指針や基準があれば効果的である。しかし、現状ではこのようなものは十分でなく単なる基本方針にとどまっている。つまり、一読して分かるように単語の綴りの略し方に規則性をもたせ、かつ曖昧にならないようにすべきである(1)とか長い変数名は#などの記号で区切る(2)といった留意事項や提案はあっても具体的な基準への展開は十分とはいえない。

具体的な基準とは、例えば変数の長さはどのぐらいにすべきであるかとか「カウンター」はどの様な名前にしたらよいか、名前には合成語と単一語があるが、この違いによって略し方をどのように変えるべきかなどを規定しているものである。現状では、このような基準はない。名前の命名はプログラマー個人に任されている。プログラマーは苦労して名前をつけているが必ずしも適切であるとはいえない。変数の名前に不要な個人差を生じソースコードを難解なものにしている一つの要因ともいえる。

の解読の重要な手がかりとなるから変数の役割やその値の意味を的確に表すような、個人差のない名前が付けられる指針や基準があればコーディングにおける苦労は軽減され、他人が読んでも分かるから保守性は改善される。このような基準は実態にあっていなければ効果的とはいえない。そこで、実際のプログラムを調べてみた。ここではこの調査結果を報告する。

2.命名法の要因

(1)命名法の分類

名前の付け方についてはまだ、明確な分類は確立していない。ここでは論を進める都合から便宜的に、①符号方式、②連番方式、③連想方式の三種類に分ける。符号方式は名前の各桁がコード体系をなすもので桁毎に特定の意味をもつような方式である。例えば、一桁目はプログラム分類を表し、次の桁でモジュールidを表すといったものである。文字のつながりには意味をなさない。連番方式はカウンターとかワークなどのように同じ用途をもつ変数に1,2とかA,AAとかの番号を与えて識別する方式である。連想方式は文字のつながりによって特定の意味を連想させる方式である。これらの方式の主要な用途は、符号方式は管理用であり、連番・連想方式はコーディング用である。以上を表1に整理する。

本報告での分析では、連想方式での名前の付け方を対象とする。

(2)命名のプロセス

連想方式における変数の命名プロセスをモデル化してみる。まず、変数の役割をイメージとして頭の中に描く。次に、この役割のイメージを日本語の文で表し、そのキーとなる単語を選ぶ。単語の選び方には単一の単語の場合といくつかの単語を合成語として選ぶ場合がある。こうして選んだ単語を英訳するか、ローマ字

先に述べたように変数はソースコードで表し，この綴りを短縮して変数の名前とする。例えば、「部品の使用回数をもつ」変数に名前を付ける場合を想定する。数の勘定はカウンターという概念がある。そこで、「部品の使用回数を数えるカウンター」という日本語の文が決まる．これを名詞の合成語「部品使用回数カウンター」にする。代表語として例えば、部品とカウンターを選ぶと PARTS COUNTER という英単語の合成語ができる。この綴りを略して例えば、PCTRという変数名ができあがる。図1にこのプロセスを示す。

(3)名前の構成要素

変数の名前は、英文字のみからなる単語の省略名、序数（1,2,3..;A,B,C,..）、識別子（AA,AAA,..）、句切り記号（_、-、・、#、$...）から構成される。単語の省略名を連ねた変数名を部分化した変数名という。変数名および部分化した変数名の構成規則を構文図で書けば図2のようになる。

(4)単語綴りの短縮化要因

単語を短縮するにあたっては、①曖昧でない、②意味が分かりやすい、③簡潔である、の要件を満たさねばならない。これらの要件を満たすための要因を短縮化要因という。この要因は四つある。これらは、①単語の選択、②単語の長さ、③略し方（単語の綴り字のどの字を残すか）、④日本語か英語か（ローマ字綴りか英語綴りか）の選択、である。これらの要件と要因の間には関係がある。例えば、曖昧さは単語の長さと略し方に関係する。このような関係を表2に整理する。

(5)単語の分類

変数名の基になっている単語はそのプログラムに関係する分野の用語に因んでいる。関連する分野とは問題、プログラム処理、装置、その操作法、OS、コンパイラーである（図3）。ここで、プログラム処理用語とはチェックや区切りが、装置用語とはカーソルや色が、操作法用語とはメニューやコマンドが、OS用語とはファイルやディレクトリーが、コンパイラー用語とはBITやCELLが、それぞれの代表的な例である。

3.調査

3.1調査方法

(1)調査の目的

ソースコードの変数名を分析することによってプログラマーが暗黙のうちに用いている変数の命名法を推定する。

(2)調査項目

表2の、短縮化要因に添って調査すべき項目を細分化する。基本的には，変数名を構成している単語と単語を合成した合成名に分けて考える。単語については、種類、略し方、長さ、言語表現を、合成名については、長さ、合成に使う単語数、合成語の曖昧さの程度を取り上げる（表3)。

(3)調査の対象

①ソースプログラム

表4に示す。

②対象とする名前

変数のみとする。モジュール名は対象外とする。モジュール名では符号方式による命名部分の占める割合が多いので外す。

(4)調査の進め方

基本的には次の順序による。

step1.ソースコードから変数名を取り出す。

step2.取り出した変数命名の対象を絞る。
..モジュール名、同一の名前を除く。

step3.変数を部分に分け部分変数を取り出す。
..句切り記号で部分化し、同一の名前を除く。

step4.部分化した変数名の単語構成を調べる。

step5.分類し、集計する。

3.2調査の結果

3.2.1単語について

(1)単語の種類

単語の種類が多ければ標準化は難行が予想される。少なければ比較的容易である。表5にソースコードから取り出した変数名と更にこれから取り出した単語の種類を示す。表中の標準化単語とは同一の単語であっても略し方の異なるものがある。これを一つにまとめたものである。つまり、BUFFERをBUF、BFR、BFなどのように略し方が異なるものをひとつとして数えたものである。このような略し方の異なるものを後述するように略し方のバリエーションという。定義されている変数名数の約1/19、同一の名前を除いたものに対しても約1/7である。

(2)単語の分野

単語が図3での分野のどこに属するかを図4に示す。この結果によればプログラム処理の単語が42%を占める。今回は操作法に関するプログラムなので操作法の単語が23%を占めているがこの比率はプログラム分野によって変わると予想される。

(3)略し方

単語の綴りを簡潔にかつ曖昧にならないように略す方法を探るために現状で使われている略し方とそれらの曖昧さ(他の単語との衝突)を調べる。

■略し方と語長の関係

略し方の分類とこの出現頻度を表6に示す。表6の分類は必ずしも直交していない.しかしここでは一意になるように分類した。先頭からn文字取り出す略し方と先頭と尾部を取り出す略し方で全体の約3/4を占めている。

■略し方のバリエーション

同一の単語の異なる略し方をバリエーションという。この頻度を図5に示す。バリエーションのないものは57%で、最大で六つもつものがある。

■衝突

略したものが別の単語と衝突すると解釈に曖昧さを生ずる。表7に語長と衝突の関係を示す。語長が短いと衝突の可能性も高い。長さが1では11個の単語と衝突する場合が3件ある。長さが4以上では衝突する場合はない。

(4)言語表現

ローマ字は8.1%で残りは英単語の綴りである。因みにミススペルは表　に示すように2%ある。

3.2.2合成語の名前

(1)変数名の長さ

変数名の文字数を長さという。長さの分布を図6に示す。7が最も多く、6～8に集中している。この三つで83%を占める。なお、9以上の長さのものがない理由は言語仕様上の制限からである。

(2)合成に使っている単語数

部分化された合成名が幾つの単語によって合成されているかを表8に示す。部分化した合成名の半数(51%)は合成名でなく単一の単語である。二語から合成されている名前が41%を占める。三語以上は8.2%である。部分化した変数名で長さ1の単語を含むものは合成語のうち、92%である。つまり、長さ1の名前は合成語用の略し方といえる。

(3)区切れる可能性の数

部分化した合成名の区切り方は一通りではない。長さnの名前は、2^{n-1}通りの区切り方ができる。句切りが妥当か否かは、区切った単語が意味をなすかどうかによる。例えば、CHNGはCHaracter NGともCHaNGeとも解釈できる。区切りの可能性の数は少ない方が望ましい。対案が多ければ分かりにくい要因になる。長さと区切りの可能性の数の関係を表9に示す。可能性が2または3のものが88%に達する。長さが3で区切りの可能性が2のものは31%である。

3.3考察

(1)綴りを略してない単語の長さ

綴りを略さないで変数の名前にすれ

ばコードは見やすくなる。特に他人のコードを読む場合には効果的である。この反面、書いたりキー入力するのに手間がかかるとか、見慣れたコードではかえって煩わしいとも言われている。今回の調査では、綴りを略さずに変数名とするものは全体の8.7%(表6)、長さが4以下のものが92%であった。つまり、現状では綴りを省略しないでそのまま変数名としているのは長さが4以下の単語であるといえる。

(2)よく使われる略し方

頭から何文字かをとる方法が全体の50%である。頭部と尾部をとる方法は24%である。単語の略し方では前者の方法には合成語で使われる先頭の一文字だけをとるものも含まれているのでこれ(25%)を除くと両者はほぼ同程度であり、合わせて半数である。

(3)ミススペル

「思い付き」の略し方に分類されるものには頭部と適当な子音字をとる方法、N(oNline), X(eXchange)のように子音字のみをとる方法、およびBUF BLLFと誤るようなキー入力ミスやミススペルが含まれている。思い付きの分類のうち最後のミススペル等が6割を占める。誤ったままでもプログラム動作には影響がないので通常は放置されている。

(4)変数名の分かりにくい理由

■合成語

名前に変数の用途などを正確に表そうとすると多くの修飾が必要となる。例えば、ファイル名を例にすると、名前、入出力の区別、表現形式(文字/16進)が名前を付けるときの主な要素となる。これらの単語を略さないでつなげると例えば、FILE_NAME.INPUT.HEXADECIMALのように長くなる。これを嫌って略すので分かりにくい名前になる。

■区切りがない

区切り記号をもたない合成語はいろいろと区切れる。表9によれば、区切れる可能性が3以上あるものが45%を占める。

■略し方が様々である

単語の約4割は1～3通りの異なる略し方がある(図5)。

■綴りがローマ字か英語か分かりにくい

4.結論

今回の調査分析からプログラマーが暗黙にもっている変数の命名法は次のように推定することができる。

(1)変数を構成している最小の単位となる名前は、単語の綴りの頭から3～4文字とるか、頭部と尾部を併せて3～4文字とる。

また、長さが3～4文字の単語は省略しないこともある。

(2)合成語にするときは一文字と三文字を組み合わせた名前を単位としてさらに合成する。全体の長さは6～8文字である。

(3)240種類の単語を使っている。

(4)単語の約4割は1～3通りの異なる略し方がある。

以上

参考文献

1.Christine N. Ausnit, et al.: Ada in Practice, Springer-Verlag(1984) ISBN 0-387-96051-1

2.佐藤、稲田、浅見:保守用ドキュメントを指向したプログラム記述法、情報処理学会第21回全国大会(1980)

表　1　　命名法の分類

用途	命名方式	概　　要	例
管理	符号方式	桁構成にコード体系を重ねる。 桁毎に特定の意味をもつ。	AM.. ↑↑モジュールid └プログラム種別
コーディング	連番方式	同じ用途を番号で識別する。	WK1,WK2
	連想方式	文字のつながりで特定の意味を連想させる。	STK ..スタック

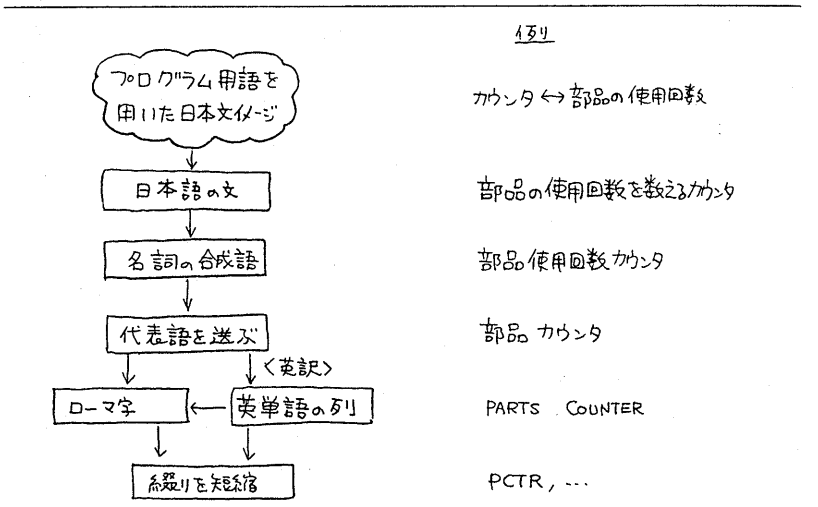

図1　命名のプロセス（連想方式の命名法）

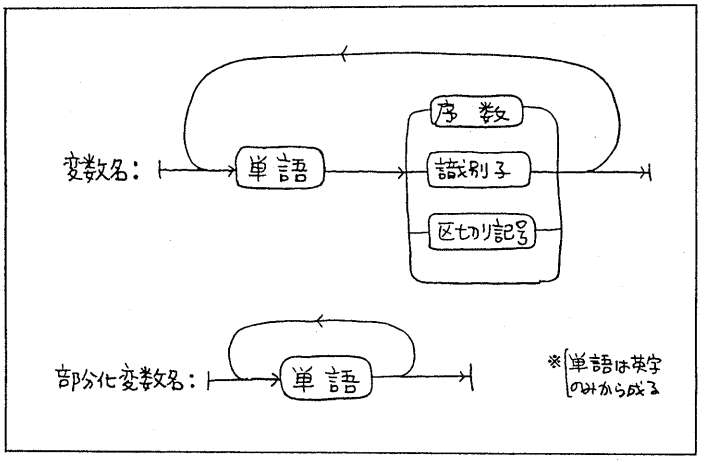

図2　変数名の構造

表　2　　短縮の要因

要　件		要　因
曖昧でない		単語の種類
意味が分かりやすい		語の長さ
簡潔である		略し方
		日本語，英語の選択

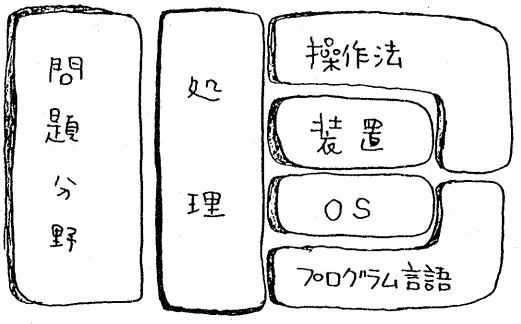

図3　単語分野の分類

表　3　　調査項目一覧

単語について	種類 略し方 長さ 言語表現
合成語 について	長さ 構成単語の数 合成の仕方

表　4　　ソースプログラムの概要

項目	内容	備考
機能種別	DB検索・編集	
モジュール数	39	コンパイル単位の数
規　模	11.4〔千行〕	開発要員、延べ5名
記述言語	SYSL-S	PL/Iのサブセット

表　5　　正規化した単語の数

名　前	内　容	出現数
検出した変数名	ソースコードから取り出せた変数の名前	4547
対象とする変数名	同一の名前、モジュールの名前を除く	1646
部分化した変数名	区切り記号で区切り、同一名を除く	531
単　語	部分化した変数名を意味上から区切り、同一の名前を除く。	437
正規化した単語	略し化の異なるものを一つにする。	237

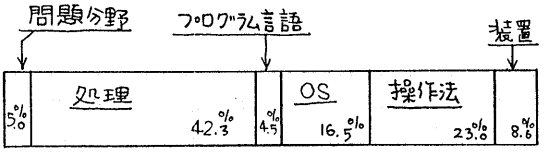

図4　分野ごとの出現頻度

表　6　　略し方と頻度

略し方	例(ONLINE)	1	2	3	4	5	計
頭からｎ字	ONL	100	24	78	16		218
頭部と尾部のみ	ONE		27	75	4		106
頭部と子音	OLN	8	11	6	2		27
慣用	ON		5	12	16		33
思い付き	N	3(1)*	3(2)	6(4)	2(1)	1(1)	15(9)
省略しない	ONLINE		2	13	20	3	38
合　計		111	72	190	60	4	437

:ミススペル分の再掲

表　7　　略し方の衝突

長さ	衝突の数 2	3	4	5	6	7	8	9	10	11	合計
1	6	3	1	4	1	1	1		1	3	21
2	7										7
3	8	1									9

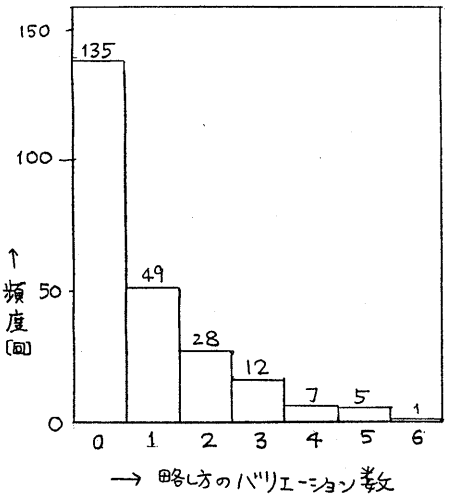

図5　略し方のバリエーション数と頻度

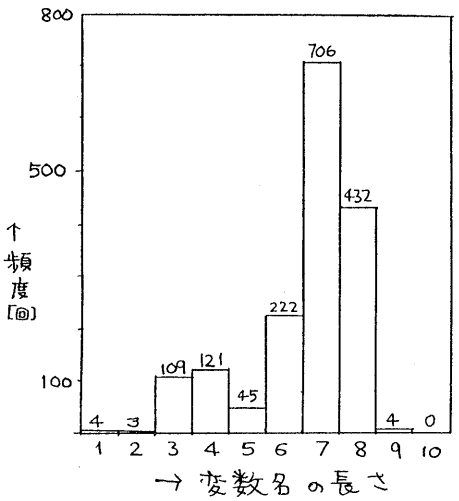

図6　変数名の長さの分布

表　9　合成語の区切り方の可能性

長さ	最大	可能な区切り方 1	2	3	4	5	6	7～	合計
2	2	2.9	7.5	0.2					10.5
3	4	2.5	31.0	14.5	1.7				49.7
4	8	0.7	12.0	12.4	5.9	2.5	1.1		34.6
5	16		0.2	0.4	1.5		0.2	0.6	2.9
6	32					0.2	0.4	0.7	1.3
7	64							0.2	0.2
8	128							1.0	1.0

表　8　　部分化された合成語の区切り方と分割数

長さ	区切り方	分割数 1	2	3	4	5	6
1	1	13					
2	2	39					
	1-1		16				
3	3	162					
	2-1		19				
	1-2		66				
	1-1-1			9			
4	4	54					
	1-3		55				
	3-1		33				
	2-2		12				
	1-1-2			14			
	1-2-1			4			
	2-1-1			2			
	1-1-1-1				5		
5	5	3					
	3-2		3				
	2-3		2				
	4-1		3				
	1-4		2				
6	3-3		2				
	4-2		1				
	1-2-3			3			
7	2-5		1				
	4-3		1				
	3-1-3			1			
8	2-1-1- -1-1-2						6
合　計		271	216	33	5		6